Certificate of Mailing Under 37 C.F.R. 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as Express Mail in an envelope addressed to: **BOX PATENT APPLICATION**, Assistant Commissioner for Patents, Washington, DC 20231 on:

Date: November 20, 2001 Express Mailing Label No.: EF384080865US

Signature: Crystal Slason

Typed or Printed Name: Crystal Slason

Inventors: Vincent E. Parla, Peter J. Moss, and Kevin J. Bergquist

Attorney Docket No.: CIS01-06(4183)

METHODS AND APPARATUS FOR EVENT HANDLING

BACKGROUND OF THE INVENTION

Computers and computer networks are often multi-component systems that are subject to a variety of hardware and software events such as errors or failures of different origins. In such systems, it is often desirable to be able to monitor or collect events that occur in order to be able to diagnose problems associated with systems. In particular, many computer network systems may be very complex and may include both hardware and software components for which events are to be monitored. As an example, the computer systems and network components may be critical to important operational functions of high reliability systems. These and other situations have resulted in the development of tools and techniques for monitoring events such as system or component failures. Two of such conventional mechanisms for monitoring events are the simple network management protocol (SNMP) and phone home event reporting systems.

The simple network management protocol (SNMP) is a software system and protocol that uses SNMP agent software operating on one or more computerized devices to identify certain computer and/or network related events and report these events to a network management server computer system operated by support technicians or other individuals for problem resolution. In particular, SNMP can report hardware and software events such as errors, problems, and pre-determined operational characteristics, etc. to the management server. The SNMP agent runs on each system component that is to be monitored in order to collect data regarding that system component. An SNMP server collects data provided by the agents about the computer systems, networks, software, etc. being monitored and can provide simple commands back to the SNMP agent in order to adjust operational characteristics of the system component. The data to be collected is identified and stored in a management information base (MIB).

Another system for reporting hardware and software events, is referred to as a "phone home" system. This type of system allows a computer system experiencing a problem to dial out over a telephone line to a support server in order to communicate event-related information regarding the problem. The support server can be monitored by support personnel in order to diagnose the problem reported using the phone home technique.

SUMMARY OF THE INVENTION

Unfortunately, there are deficiencies in the conventional techniques and mechanisms for managing system problems. Due to the complexity of computer and network systems neither SNMP nor "phone home" provide adequate tools for monitoring events. Although SNMP can identify and report problems and events, it has limited capability for interpreting or analyzing specific events or problems. SNMP does not provide the ability to identify or analyze the hierarchical relationships between events or maintain and use state information. Nor does SNMP have the ability to provide operators monitoring SNMP reports with any advice or assistance about actual

problem resolution. Most of the analysis of problems and events that is necessary for diagnosis and resolution of problems must be undertaken by operators on a manual basis. Furthermore, conventional techniques and mechanisms for reporting errors or event information are ineffective at taking full advantage of the opportunity to use events with similar or related causes as a diagnostic tool. In other words, conventional error reporting systems cannot correlate different events in order, for example, to deduce the origination of a specific problem or event.

Another disadvantage of conventional techniques and mechanisms for event reporting is that existing systems operate as closed systems that are able to respond only to system events that have been pre-defined and/or pre-programmed into the event sending and receiving components of the system (e.g., an SNMP agent and server, or a phone home system). This inflexibility, makes it difficult to support error reporting for existing systems at the same time as a new system or systems for which enhancements are being implemented. For example, if a new version of a software program is installed that uses a different error message or error messages than the error message or messages already identified to an SNMP agent and server, the management information base (MIB) of the SNMP system will have to be upgraded before the system would be able to respond to any new error message. Unfortunately it is a fairly manual and complex task to keep the MIB up-to-date each time that a new system component is added or upgraded. This inflexibility of a conventional event handling system is particularly cumbersome when users or customers elect to obtain their technical support and event handling from third party, centralized, automated or semi-automated, or off-site system and event support facilities.

Another disadvantage of conventional mechanisms and techniques with respect to users of third party, centralized, automated or semi-automated, or off-site event support facilities is that in such arrangements, it may be difficult or impossible to provide a communications channel between the host and/or supported systems, the event information component (e.g., MIB or management information base in the case of

SNMP), event processing server (e.g., support server) and the output device (e.g., monitoring agent), and the like if a shared communications channel (e.g., such as a local area network, wide area network, etc.) does not exist between those system components or if the event monitoring system is not designed to operate using the communications protocols with which the user's system is equipped.

Conventional systems that have the ability to dial-up or "phone home" to an event processing server (e.g., support server) whenever a problem, error message or other event must be reported require a dedicated phone line and must be specifically set-up and configured for the specific location, communications facilities available, telephone number, etc. In addition, the initial database of the customer facilities, systems, devices, etc. must be predefined prior to event monitoring taking place.

Furthermore, event monitoring tools may be unable to monitor events in computer systems and/or networks operating within certain multiple-platform environments. For example, a company that is using a Windows-based SNMP monitoring system may not be able to monitor certain Linux®-based system events within the Windows SNMP system due to the use of nonstandard MIB's.

Embodiments of the invention are directed to techniques and mechanisms that provide greater flexibility and additional capabilities for event monitoring, reporting and handling and significantly overcome these and other problems of conventional error or event reporting systems. In embodiments of the invention, event information (e.g., company, product, module, help, event ID, message mapping, substitution mappings, hierarchical, chart templates, rules, etc., such as XML templates) is maintained within a storage facility connected to an event processing server and is used to process event data. Sources of event data (e.g., hardware or software programs generating a hardware error message, alarm events, system status message, software program failure report message, etc.) can produce events which are detected by an event generation process operating in conjunction with the source of event data. The event generation process can communicate with the event processing server to report the event, such as by using

XML to transmit the event data to the event processing server. Upon receipt of event data notifying the event processing server of the occurrence of such an event, the event processing server can process the event data using stored event information (e.g., XML documents) to determine how to interpret, manage or handle the event data.

As part of developing a new product, the product developer integrates event information into the product. When the product is put into service, the product automatically sends event information to the event processing server updating the existing event information. Therefore no advance knowledge or manual interaction is required. Although the initial installation of the system may include an initial installation of event information, additional event information may be dynamically added to the storage facility of event information. When the event generation process begins operation of monitoring event sources for event data, the event generation process can perform a registration operation with the event processing server. During the registration process, event generation process can provide registration information to the event processing server that indicates, for example, a specific company, product, module (e.g., software module) or other information that identifies a source or sources of forthcoming event data as well as event information that the event processing server will require in order to be able to correctly process the forthcoming event data. In other words, the event generation client can pre-register itself with the event processing server to indicate which product areas events will be reported within and what event information the event processing server will require to be able to properly interpret the event data reported in those product areas. Upon receiving event registration information, the event processing server can perform a check to ensure that the event information identified within the event registration information is available for access by the event processing server. If the event processing server determines that it does not currently have access to the event information identified in the event registration information, the event processing server can provide an event rejection back to the event generation process indicating any missing event information that the event processing

server does not currently have access to. In response, the event generation process can provide the missing event information back to the event processing server in the form of event information documents such as XML documents. In this manner, embodiments of the invention allow an event generation process to specify what event information the event processing server will require in order to properly process event data. Accordingly, the event processing server can dynamically "learn" how to process events as they are generated without the need to manually update a management information base or other database.

Alternatively, in other embodiments, no pre-registration is required. In such cases, the event generation process can begin reporting event data to the event processing server upon detection of the first event without first registering with the event processing server. Upon receipt of event data by the event processing server, if event information required by the event processing server to process that event data has not been provided in advance of receiving the event data (e.g., notifying the event processing server of occurrence of an event), the event processing server will return an event rejection. In other words, if the event processing server is missing event information required for properly processing the event data (i.e., such missing event information that may be identified along with providing the event data to the event processing server), the event processing server immediately returns an event rejection notifying the event generation client which event information it should provide in order to complete event data processing. In this manner, if the event processing server detects an event which has never occurred before, for example, from an obscure product area, the event generation process (e.g of the event generation client) can report the event in an event data message along with an indication of event information required to process this event data. If the event processing server does not have this necessary event information, the event processing server can use the event rejection technique to indicate that the event generation process must supply the server with the missing event information so that the event data can be properly processed.

In one embodiment of the invention, events are managed as collections of related events that are treated as objects and tracked by the event generation client. The event processing server manages and tracks the hierarchy of the objects. The hierarchy of objects can be defined by templates. Accordingly, the event processing server, by processing objects according to information provided in the templates, can identify problems or other issues based upon the interrelationships between multiple objects.

One embodiment employs the use of extensible mark-up language (XML) format documents as the communications medium for information exchange between the components of the system. Accordingly, product developers are provided with XML templates that can be used to format the various components of information used (e.g., event registration information , event information, event data) which the event generation client and event processing server are configured to generate and process. Typically, sources of event data such as application software programs, embedded software and other software which control hardware are programmed to provide event data (e.g., including, for example, company, product, module, help, event ID, etc.) or generate events which an event generation process can detect and report to the event processing server using the format provided by the XML templates. In addition, customers can generate event information templates of their own in order to accommodate their unique event handling requirements.

Since the event processing server can process text-based XML data, transmitted for example, by means of the commonly available and widespread HTTP protocol (as described earlier), the embodiments of the invention provide a mechanism for accepting event information and event data to support systems operating on a variety of platforms. For example, it would be feasible to simultaneously accept and process event information and event data from event generation clients operating on various operating system platforms such as Microsoft® Windows®, Solaris®, Unix®, and so forth with one version of an event processing server. In addition, embodiments of the invention provide a mechanism for tunneling SNMP over HTTP and the ability to automatically

forward the MIB to the event processing server.

In particular, an event processing server, in one embodiment of the invention, provides a method for processing events comprising the steps of receiving event registration information, identifying event information required to process event data, and based on the event information, determining if existing event information is accessible to process the event data. If the existing event information is not accessible the event processing server provides an event rejection indicating (i.e., to the event generation client) missing event information and then receives the missing event information identified in the event rejection. During this process, if the existing event information is accessible to the event processing server (i.e., no event rejection required or the missing event information provided in response to a former event rejection provides all of the required event information needed by the event processing server), then the event processing server provides an event data destination (e.g., a URL) to the event generation client and begins to receive the event data via the event data destination.

In another embodiment, the event registration information includes event registration information having unique identifiers identifying the source of the event data and event information required to process the event data. As an example of such an embodiment, the event registration information can contain unique identifiers that can identify such things as a product source, company source, software module or routine source or other source of event data as well as unique identifiers identifying event information such as the necessary XML templates or documents that will be required to process event data generated from the sources.

In another embodiment, event information (i.e., the event information used to process event data) is selected based on the event data received and the system generates an event output from the selected event information. As an example, event data may be received in the form of an XML-based message that indicates a simple event code and an associated text message to be used in conjunction with event information XML

documents. The event processing server can use the event code to traverse the set of event information available to the event processing server in order to arrive in a location within the event information (referred to herein as the selected event information) that is related to the event code specified in the event data. Using the XML definitions defined at that location in event information, the event processing server can process the event code and the associated event text message to produce event output.

In still another embodiment, the event processing server accepts at least one of event registration information, event data and event information mark-up language documents. In other words, the event processing server can accept any of these types of documents and can process them according to the embodiments of the invention as explained herein.

In another embodiment, the event data includes network management data (e.g. SNMP data) indicating a network management event associated with a source of the event data, wherein the step of receiving event data utilizes hypertext transport protocol to receive the event data. Accordingly, this embodiment of the invention relates to processing events related to network management issues, SNMP tunneling and MIB forwarding.

In another embodiment, the event processing server reads first and second event data and processes the first and second event data to produce event output data that reflects a hierarchical event relationship between the first and second event data. This may be done, for example, using the interrelationships expressed in the various event information documents such as XML documents. As an example, the first event data may be event data identifying an event in a specific product or within a specific software routine within that product. The second event may identify another event within a different product area. However, since the event information used by the event processing server is preferably an arrangement of XML documents which can cross-reference and interrelate to one another, production of event output may require the appearance, occurrence or existence of both the first and second event data in order to

trigger the event processing server to reference selected event information in order to produce an event output which reflects an error condition related to the first and second event data. It may be, for example, that the two products each experiencing independent events require each other's proper operation in order to function correctly. Accordingly, the first and second event data is may indicate to the event processing server, through the event information definitions, that when each product experiences its respective event data (i.e., the first and second events), the event output may indicate that the combined operation of each product is non-operational. In other words, the existence of the first and second event data from totally independent sources causes an escalated event to occur and to be reported in event output. To provide a more specific example, that is, the case of a router and switch, a router error, might, by itself, not be a fatal error. However, the same error, occurring in conjunction with a related switch error, on a separate system component, could indicate that a WAN/LAN connection has failed.

In yet another embodiment, the event processor creates system component status records and updates a status of the system component status record based on the event data received. In this manner, the event processing system of this embodiment of the invention is able to provide a status reporting mechanism for periodic status reporting concerning event sources.

Other embodiments of the invention relate to the event generation client. In one such embodiment, the event generation client participates in the event registration processing previously discussed by providing, to the event processing server, event registration information that identifies event information that the event processing server requires to process event data. In response to providing event registration information, the event generation client can receive an event rejection specifying missing event information that the event processing server does not currently possess. In response, the event generation client (e.g. or client process) can provide the missing event information to the event processing server so that the event processing server can properly process event data. Also in this embodiment, the event generation process or

client detects an event, and in response to detecting the event, creates event data and sends the event data to the event processing server. The event processing server utilizes event information which the event processing server currently possesses to process event data to produce event output.

In another embodiment, the event generation client, in creating event data, formats the event data in a mark-up language format capable of transmission via a hyper-text transport protocol. As an example, the event data may be formatted in XML format.

In yet another embodiment, the event generating client initiates a multiple of status checks of sources to produce status check information and forwards status check information in the event data to the event processing server. In this manner, the event generation client can provide a heartbeat or other periodic status record to the event processing server regarding the status of event sources.

In still another embodiment, the event generating client periodically sends event data to the event processing server as confirmation of an operating communications channel.

In yet another embodiment, the event generation client receives an event rejection indicating missing event information from an event process server, then obtains the missing event information and sends the missing event information to the event processing server. In this manner, the registration process of the event generation client ensures that the event processing server has all of the required event information (i.e., XML documents) that are necessary to properly process event data.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular description of preferred embodiments of the invention, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not

necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

Fig. 1 shows an event handling system with components that are configured according to one embodiment of the invention.

Fig. 2 is a flow chart of a procedure which is performed by the event processing server according to one embodiment of the invention.

Fig. 3 is a flow chart of a procedure which is performed by the event generation client according to one embodiment of the invention.

Fig. 4 shows an example of the event handling system in operation and which illustrates operation of example embodiments of the invention.

Figs. 5-7 are a flow chart of a procedure performed in an example of the event handling system configured according to one embodiment of the invention.

DETAILED DESCRIPTION

The invention is directed to techniques and mechanisms for event monitoring, reporting and handling. In a hypothetical implementation of the invention, customers have hardware, software, and/or systems that are in need of monitoring. The hardware, software, and/or systems, referred to herein as event sources, communicate with an event generation client that can, in turn, communicate the occurrence of such events to an event processing server. The event processing server processes the event-related information, in an appropriate manner, in some cases providing analysis and/or, in some cases providing reports for technical support personnel to monitor and view at a remote location. In particular, related events are treated, that is, managed and tracked, etc. as objects. The condition of the related events may be observed and used to draw conclusions about the objects (e.g. observed events such as certain events being "down" may be used to draw conclusions about the particular objects, such as, that the state of a particular object is impaired or down.). In turn, objects can also be related to one another, following a logic spelled out within event information (e.g. event information

that was formatted according to templates).

For example, three different events 10, 52, and 25 are used collectively to identify an object called object x. The different events relate to completely separate devices. Object x is set up to apply logic to the events such that if both events 10 and 52 are down (e.g. inoperable) the object state is defined as impaired. On the other hand, if all three events (e.g. 10, 52, and 25) are down, then the object state is defined as down. Additional logic can be applied to objects, as well, whereby object states can be "rolled-up" into other objects. For example, two object, object x and object y are defined to roll-up in a hierarchical fashion into a third object, object z. Logic for the objects is defined such that if object x or object y is down (e.g. inoperable) then object z is defined as impaired, but if both objects x and y are down, then object z is defined as down.

Accordingly, in one embodiment, event information (e.g., information related to one or more of a company, product, module, help, event ID, message mapping, substitution mappings, hierarchical, chart templates, etc.) is maintained within a storage facility connected to an event processing server. The event information is expressed in one embodiment as a set of XML template documents that dictate or describe how event data is to be processed by the event processing server. Upon the initiation of the system, such as upon systems start up or upon detecting an initial event, the event generation client sends registration information to the event processing server to make it aware of event sources and event information that will be required to process event data (e.g. information about software and/or hardware occurrences) that will be sent from the event generation process to the event processing server. The registration information indicates to the event processing server to specific source of forthcoming event data as well as the complete set of event information that will be required to process event data once received. By complete set of event information, what is meant is that the event registration information in one embodiment specifies a set of one or more event information templates, such as XML documents, that the event processing server must

possess in order to properly process the forthcoming event data. In this manner, embodiments of the invention allow an event generation client to preregister within the event processing server to give the event processing server a "heads up" as to what will be required to properly handle and process event data which the event generation client will begin to report to the event processing server.

When an event occurs within an event source (e.g., hardware or software program generating a hardware error message, system status message, software program failure report message, etc. all within the customer facility) the occurrence of the event is communicated to an event generation client, which, in turn, then generates event data which is communicated to the event processing server. Upon receipt of event data (e.g., notifying the event processing server of the occurrence of such an event), the event processing server uses stored event information (e.g., stored information about how to process the event data, such as the event information retrieved during the registration process) to determine how to interpret or handle the event data. If the event processing server has insufficient event information to process the event data received, it can send an event rejection to the event generation client notifying the event generation client as to which additional or missing event information is needed to process the event data. The client can respond by providing the missing event information. In response, the event processing server can provide the event generation client with an event data destination such as a URL indicating where the event generation client is to begin sending event data. In some embodiments, event rejections can be used even after registration is complete in order to indicate to the event generation client that the event processing server is lacking the necessary event information to properly process event data for a specific event.

One embodiment of the invention employs the use of extensible mark-up language (XML) format documents as the communications medium for information exchanged between the components of the system. Accordingly, product developers of the system are provided with XML templates or schema that can be used to format the

various components of information used (e.g., event registration information, event information, event data). As described earlier, the templates also provide the event processing server with information about the hierarchical relationships between different objects. Typically, sources of event data such as application software programs, embedded software and software programs which control hardware will be programmed to provide event data (e.g., including, for example, company, product, module, help, event ID, etc.) to the event generation client using the format provided by the XML templates. In addition, customers can generate event information templates of their own in order to accommodate their unique event handling requirements. Such templates can inherit from base templates if so desired.

Fig. 1 shows an event handling system 100 which is suitable for use by the invention. The event handling system includes an event processing server 110, an event generation client 210, a network 105 and an event source 142. In addition, the event processing server 110 is connected to a storage device that stores event information 124, such as a disk drive or other storage facility.

The event processing server 110 is also connected to an output device 114, such as a simple display mechanism or a more complex system capable of evaluating, formatting, presenting, compiling, and/or organizing of event data 134, event information 124, etc. The event handling system 100 may be configured with multiple event generation clients 210, all capable of generating event messages 140.

The example system shown in Fig. 1 uses a series of event messages 140 transmitted over the network 105 between the event generation client 210 and the event processing server 110 including event registration information 130, event information 124 and event data 134. Also included are event rejections 136 and event data destinations 138.

An event source or sources 142 can include one or more software and/or hardware devices. It should be understood that there can be many event sources 142 (e.g., sources of events) coupled via a network 105 or other communications medium

which operate according to embodiments of the invention. Some examples of event sources are software programs that are programmed with the ability to report events such as error messages, information about the software operation, etc., hardware or other devices capable of reporting error messages, problems, failures, etc.

Event sources 142 may communicate with event generation clients 210 over the network 105 or by other types of connection. In other cases, the event source 142 and the event generation client 210 may function integrally as single unit, such as, for example, a software program programmed to perform both the software program's normal operating functions and the functions of an event generation client 210. A hardware device may also have firmware, software or other programming which enables the device to perform the functions of an event generation client 210 in addition to performing its normal hardware functions. For example, a card, device or other hardware component of a computer system will usually be controlled by software or software-encoded firmware or electronics, etc. Another example is a router, which is encoded with software performing both the router's operational functions as well as the event generation client functions.

The event processing server 110 includes a communications interface 115, a memory 112 and a processor 113. It also includes an interconnection mechanism 111 (e.g., a data bus and/or circuitry) which couples the communications interface 115 (e.g., modem or other network interface), the memory 112 (e.g., any computer readable medium such as a random access memory (RAM) and/or read only memory (ROM) or even disk or storage medium), the processor 113 (microprocessor or central processing unit), the communications interface 115, the storage device for storing event information 124 and the output device 114. The memory 112 can store (i.e., is encoded with) an event handler application 120-1 software program. The processor 113 can operate an event handling process 120-2, which represents execution or other performance of logic instructions that form the event handler application 120-1 or other software. The communications interface 115 allows the event processing server 110 to

communicate with the event generation client 210 and other components of the system via the network 105.

The event generation client 210 has a communications interface 215, a memory 212 and a processor 213. It also has an interconnection mechanism 211 (e.g., a data bus and/or circuitry) which couples the communications interface 215 (e.g., modem or other network interface), the memory 212 (e.g., any computer readable medium such as a random access memory (RAM) and/or read only memory (ROM) or even disk or storage medium) and the processor 213 (microprocessor or central processing unit) and communications interface 215. The memory 112 can store an event handling process 220-2, which represents execution or other performance of logic instructions that form the event generation application 220-1 or other software. The processor 213 can operate an event generation process 220-2 and/or other process which can either be the event generation application or other software. The communications interface 215 allows the event generation client 210 to communicate with the event processing server 210 and other components of the system such as the event source 142 via the network 105.

Fig. 1 shows one event generation client 210 for ease of description of the invention. It should be understood that there can be multiple event generation clients 210 coupled via the network 105 that each operate according to the embodiments of the invention.

The event processing server 110 and event generation client 210 may be any type of device or system such as digital or electronic devices, personal computers, workstations, servers, networked systems, or lager mini-computer or mainframe systems, or the like.

The components of the system 100 provide the methods and techniques for an event generation client 210 to generate and transmit event data 134 to a event processing server 110 for processing with the use of event information 124. Further details of the invention will now be provided with reference to Fig. 2.

Fig. 2 is a flow chart of a procedure performed by the event processing server

according to one embodiment of the invention.

In step 310, the event processing server 110 receives an event message 140 (e.g., an error message from the event processing client 110 that detects a failed process or event registration information 130 from the event generation client 210, for example, including at least one unique identifier identifying the source of event data 134). In a preferred embodiment of the invention, the first event message 140 received by the event processing server 110 is an event registration information message 130. In another embodiment of the invention, however, the first event message 140 received by the event processing server 110 may be an event data message 140 which contains not only event data 134 but also an indication of event information 124 that will be required to process the event data 134. In such cases, the event message 140 contains both event data 134 and a specification of event information 124 (i.e., quite similar to event registration information 130) that the event processing server 110 will need to properly process the event data 134.

Accordingly, in this example embodiment, event messages 140 may be either event registration information 130, event information 124, or event data 134 or a combination thereof. Collections of related events can be defined as objects. The objects possess state characteristics and as previously describe, and can be rolled-up in a hierarchical manner. In this example, event messages 140 are provided in the form of XML (Extensible Mark-Up Language) documents. XML is a meta-markup language that can be used to define a syntax capable of identifying different parts of a document in a fashion that makes XML documents a good mechanism for exchanging data between different system components. The XML documents are used to exchange information between an event generation client 210 and the event processing server 110.

Event messages 140 are formatted according to a pre-established format defined by an event handling system developer. The event handling system (501, See Fig. 4) can distribute the document formats (e.g., format of documents containing event registration information 130, event information 124, or event data 134) to the product

developers (504, See Fig. 4) (manufacturers of the event sources, such as, for example, hardware and software developers providing software and equipment, etc.) in the form of templates (504, See Fig. 4). In turn, product developers 502 (e.g., hardware and software vendors) design their equipment and systems to generate event registration information 130, event information 124, and/or event data 134 in compliance with the defined protocols and templates 504. Event messages 140 can contain globally unique identifiers (GUID's) (e.g., See 518, Fig. 4), that is, identifiers which use one of several well-known techniques for defining data elements that are assured to be globally unique (e.g. thereby allowing autonomy during the design phase). For example, using one of the techniques for developing a GUID (518, See Fig. 4), a designer can be assured that a company name acronym is not used to represent two different companies, or that the module of a particular software program does not use the same name for two different software modules. In this example, GUID's (518, See Fig. 4) may be used for company, product, module and template ID's that make up a part of event registration information 130, event information 124, and event data 134.

Event registration information 130 documents are documents that the event generation client 210 sends to the event processing server 110 in order to notify the event processing server 110, in advance, of event information 124 that the event processing server 110 will be required to have access to in order to process the event data 134 that it can expect to receive at some later point in time. Event information 124 documents are documents that provide required event information that the event processing server 110 needs in order to process any particular event data 134. The event data 134 represents documents that are sent to the event processing server 110 containing information about the occurrence of software and hardware reportable events.

In step 312, the event processing server 110 identifies event information 124 required to process event data 134 based on the event message 140. As discussed earlier, the event processing server 110 uses event information 124 to process event data

134. Accordingly, in step 312, the event processing server 110 can receive an event message 140 such as an event registration information message 130 and can ascertain from this message the set of required event information documents 124 that are necessary to process event data. In the alternative embodiment in which the event message 140 is an event data document 134, the event data document 134 may also specify or include a reference to (i.e., a globally unique identifier) event information 124 that the event processing server 110 will be required to access in order to properly process event data 134.

The event processing server 110 may receive event information 124 in advance of receiving event data 134 or the event processing server 110 may receive event information 124 after the event processing server 110 has notified an event generation client 210 to provide any missing event information, such as may be done in response to receiving an event registration information message 130.

In one implementation, a customer 502 can add or modify event information 124 in order to customize already existing event information 124. Event registration 130 by an event generation client 210 may be initiated in either of several different circumstances, either automatically, or based on user-issued control or a combination. For example, in one configuration, the event generation client 210 (connected to the software or hardware event source 142) is configured in such a way as to send an initial set of event information 124 to the event processing server 110, immediately upon being placed into service. Another option is for the event generation client 210 to send event registration information 130 to the event processing server 110 as a result of the occurrence of an event (e.g., immediately prior to sending event data 134) or as a result of the start of a sequence performed by the event generation process 220-2. An event generation client 210 may also be configured in such a manner as to send event registration information 130 at any time under its own control or as a result of user control (e.g. such as by a system administrator).

In step 312, the event processing server 110 identifies event information 124

required to process event data 134 based on the event message 140 (e. g. event registration information 130). The event processing server 110 is configured to identify the event information 124 that it will need in order to process later event data 134 which the event registration information 130 has notified the event processing server 110 to expect. In one embodiment, in step 312, the event processing server 110 obtains a list of GUID's from the event registration information 130 that indicate what XML event information 124 documents will be required to process forthcoming event data 134. Despite the existence of the event registration information 130, however, it is possible for event generation client 210 to send event data 134 without prior notification.

In step 314, based on event information 124 (i.e., identified in step 312), the event processing server 110 determines if existing event information 124 is accessible to process the event data 134. Accordingly, the event processing server 110 checks for available event information 124 (e.g., from a database, files stored on a disk drive, etc.), in order to determine if the event information 124 needed to process event data 134 is available.

The event processing server 110 uses a variety of techniques and mechanisms to determine if needed event information 124 is accessible, including GUID's (globally unique identifiers) (518, See Fig. 4). In such a configuration, event information 124 documents may include one or more fields containing GUID's (518, See Fig. 4) that can be used to search existing event information 124. The event processing server 110 uses the GUID's (518, See Fig. 4) to identify if the event information 124 necessary for processing event data 134 is available to the event processing server. If the necessary event information 124 is accessible to the event processing server 110, processing continue to step 322. As an example, in one embodiment, the event registration information 130 may contain a list of GUID's that reference specific event information documents 124 (each identified by a unique GUID) that are required to process forthcoming event data 134. The event handling process 120-2 can then consult the database of existing event information 124 to determine if all of the required event

information is available. If the event registration information 130 specifies an event information GUID that is not present within the event information database 124, then processing proceeds to step 316 explained below such that the event processing server 110 can notify the event generation client 210, via an event rejection 136, of any required or missing event information 124 that the event processing server 110 is lacking.

At this point, assume for this discussion that the event processing server 110 determines in step 314 at all of the required existing event information 124 is accessible in order to process forthcoming event data. As such, processing proceeds to step 322.

In step 322, the event processing server 110 provides an event data destination 138 to the event generation client 210. In this example, event data destination 138 may be an address to which event data 134 can be transmitted (e.g., a URL address to which event data 134 can be sent). Different configurations are possible for the event data destination 138 (e.g., the event processing server 110 may provide different event data destinations 138 to accommodate different situations). The event processing server 110, may, for example, provide a different event data destination 138 for different types of events (e.g. different types of problems) or for different locations, etc. In another embodiment, the event processing server 110 may provide different event data destinations 138 at different times in order to balance work load between more than one event processing server 110.

In step 324, the event processing server 110 receives the event data 134 via the event data destination 138 (e.g., a communications channel specified by the event data destination address). The event data 134 (e.g., occurrences reported by the event generation client 210) is transmitted to the event processing server 110 via the event destination 138. In addition to the alternatives described earlier for event data destination 138, the event handling system may be configured with one continuous event destination address 138 rather than for the event processing server 110 to generate and transmit different event destination addresses 138.

In step 326, the event processing server 110 selects the event information 124 based on the event data 134 received. For example, if the event processing server 110 receives an error message (e.g., event data 134) concerning a disk drive at a particular corporate client location, the event processing server 110 may select event information 124 related to the company experiencing the error message, about the particular disk drive experiencing the error message, the disk drive model (e.g., module) and/or the particular error which occurred. In turn, the event processing server 110 uses data taken from event information 124 documents to process the event data 134. The event generation client 210 is able to identify the state of objects and transmit the information to the event processing server 110. In turn, the event generation server 110 can process the object information based upon the hierarchical relationships between different objects.

In step 328, the event processing server 110 generates event output from the output device 114 using selected event information 134. As described earlier, the event output may simply display the event data or provide a more complex compilation of the event data 134 and may include other information.

In Fig. 2, processing steps 320 and 332 are optional and will be explained here for completeness.

In step 330, the event processing server 110 reads first and second event data 134. In some cases, event data 134 from different occurrences of events may have related causes. Accordingly, it is sometimes possible for the event processing server 110 to better identify the source of a problem by comparing two or more event data 134 occurrences. For example, the failure of a disk drive write procedure may result in a "file write" error message. There are numerous possible causes of such an error. For example, in some cases, if an existing file is corrupted, attempts to write a new file over the existing file may fail. In another situation, if a disk drive runs out of available free space, the attempt to write a new file may fail.

However, the existence of an independent CRC (cyclic redundancy check) error

message would rule out to the likelihood that either file corruption or lack of free space was the cause of a disk file write failure. Therefore, comparisons between different event data 134 items are able to provide additional information to help in diagnosis. Accordingly, as discussed earlier, the event processing server reads two or more portions of event data 134 for comparison from the same or different sources 142 applying logic to objects as defined in event information 124.

In step 332, the event processing server 110 processes the first and second event data 134 to produce event output data 114 that reflects a hierarchical relationship between objects of the first and second event data 134. Accordingly the event processing server 110 compares the objects of the event data 134 occurrences in order to identify relationships between event-defined objects that can be reported.

Step 316 concerns situations in which there is inadequate event information 124 to process event data 134 that has or that will be received. In step 316, the event processing server 110 provides an event rejection 136 indicating missing event information 124. As described earlier, the event processing server 110, as a result of receiving event registration information 130, may find that it lacks adequate event information 124 necessary for processing event data 134 that the event registration 130 indicates the event processing server 110 could receive. In an analogous situation, event processing server 110 may receive event data 134 (instead of event registration information 130 as in the earlier-described situation) for which it lacks adequate event information 124 needed for processing. In either case (receiving either event registration information 130 or event data 134) the event processing server 110 will generate an event rejection 136 in order to notify the event generation client 210 about the existence of missing event information 124.

After the event generation client 210 acquires the missing event information 124, and transmits the missing event information 124 back to the event processing server 110, the event processing server 110 receives the missing event information identified in the event rejection 136 from the event generation client 210. Further

details of the invention will now be provided with reference to Fig. 3.

Fig. 3 is a flow chart of a procedure 350 performed by the event generation client 210 according to one embodiment of the invention.

In step 360, the event generation client 210 sends event registration information 130 including identifying event information 130 required to process event data 134.

Users of the event handling system may reduce the number of the event rejections 136 generated by the event processing server 110 by submitting event registration information 130 to the event processing server 110 prior to submission of event data 134.

In one example, the event generation client 210 transmits event registration information 130 to the event processing server 110. The event registration information 130 may identify the company, product, module or other information indicating where forthcoming event data 134 will be generated from (e.g., identifies the source 142 of event data). In addition, the event registration 130 can specify all event information documents 134 (e.g., XML documents) that will be required by the event processing server 110 to process the forthcoming event data 134. In the example, the event generation client 210 provides a registration information 130 document identifying or indicating a product, such as, for example, a network digital environmental control device, and a list of events (e.g., error messages) which the event generation client 210 is aware may be sent to the event processing server 110, if such an event were to occur. The company, ACME and the product, Super-Control listed may be represented by a globally unique identifier (GUID) (See 518, Fig. 4).

Upon receipt of the event registration information 130, the event processing server 110 using the GUID's (See 518, Fig. 4) provided, (e.g., Acme/Super-Control) will search the event information 134 which the event processing server 110 has available in its information store 124. If the event information 124 required for processing the event data 134 received is not accessible to the event processing server 110, the event processing server 110 will send an event rejection 136 to the event generation client 210

informing the event generation client 210 of the event information 124 (e.g., the event information 124 identified by the ACME, or Super-Control GUID's) that the event processing server 110 finds missing in order to process the event data 134 received.

In step 362, an event generation client 210 detects an event 142.

In step 364, the event generation client 210, in response to detecting an event, creates event data 134 (e. g. formats the event data in a mark-up language format document capable of transmission via a hypertext transport protocol).

Because, as in this example, the event handling system uses XML documents as its medium of information exchange (e.g., event messages, etc.) any device or system having access to the Internet for communication of information using the HTTP (hypertext transfer protocol) can transmit event messages 140 over the Internet. For example, numerous personal computers having Internet access at individuals' homes, individual computers having Internet access over a corporate network, etc. which are all capable of transmitting event messages 140 formatted using XML would therefore be capable of transmitting the event data documents 134 to a remote event processing server 110 for processing.

In step 366, the event generation client 210 sends the event data 134 to an event processing server 110.

In addition to sending event data 134, as a result of detecting an actual event, the event generation client 210 also tracks objects (e.g. as a result of the occurrence of a collection of related events events). The event generation client 210 sends event data 134, identifying objects, to an event processing server 110. In turn, the event processing server 110, after receiving the event data 134, in the form of objects, from the event generation client 210, uses event information 124 (e.g. identified in the form of objects) in conjunction with event data 134, as originally defined by templates, to identify the hierarchy of the identified objects). Using the information collected and processed, in this way, the event processing server 110 can manage and track the hierarchy of objects who's state was identified by the event generation client 210. Accordingly, in one

embodiment of the invention, the event generation client 210 initiates a series of status checks of event sources 140 to produce status check information (e.g., event data 134). The event generation client 210 forwards the status check information, as event data 134, (e.g. in the form of objects) to the event processing server 110 for processing using hierarchical relationships defined in the event information 124. For example, mission critical systems may be configured in such a fashion that the event generation client 210 coupled to the mission critical systems initiates a status check of critical objects of the mission critical system, for example, every 10 minutes. As a result of the ten minute status checks, the event generation client 210 forwards event data 134 (e.g. in the form of objects) to the event processing server 110 after each status check, thereby reporting the operational status of designated aspects of the objects of the mission critical system (e.g., event source 142) operation. Upon receipt of the event data 134, the event processing server 110 uses templates (e.g. which describe the hierarchy of the objects) to process the event data.

In another embodiment of the invention, the event generation client periodically sends event data 134 to the event processing server 110 as confirmation of an operating communications channel. The purpose of this function is to provide a heartbeat or self-maintenance check of the event handling system 100 communications facilities (e.g., the network 105). Accordingly, the event processing server 110 is configured to report any missing periodic transmission of event data 134 from the event generation client 210. By reporting such an occurrence, the event processing server 110 provides notification that the communications channel between the event generation client 210 and the event processing server 110 may be compromised.

In step 368, the event generation client 210 receives the event rejection 136 indicating missing event information 124 from an event processing server 110. As described earlier, the event processing server 110, which has been configured to check its information store 124 for event information 124 sends an event rejection 136 to the event generation client 210 informing the event generation client 210 of event

information 124 that the event processing server 110 finds missing either as a result of any event registration 130 process or by receipt of actual event data 134 itself.

For example, if a customer (See 502, Fig. 4) installs a new module of an application program on a personal computer operating on a corporate network without providing event information to the event processing server 110 about the existence of the new module, when the event generation client 210 attempts to register the set of event data that it expects to send to the event processor server 110 with respect to the new module, the event processing server 110 will return an event rejection 136 to the event generation client 210 notifying the event generation client 210 that it has no documents available providing event information 124 needed to process the event data 134 for that particular module.

In step 370, the event generation client 210 obtains the missing event information 124.

In step 372, the event generation client 210 sends the missing event information 124 to the event processing server 110. Further details of the invention will now be presented with reference to Fig. 4.

Fig. 4 is a diagram depicting the flow of information in an example operation of the invention.

In general, the invention is directed to techniques and mechanisms for event processing. Event information 124-2 is maintained within a storage facility connected to an event processing server 110 and is used to process event data 134. Event data 134 is generated when an event source 142 (e.g., hardware or software program generating a hardware error message, system status message, software program failure report message, etc.) communicates with the event processing server 110 via an event generation process 220 to report the event. Upon receipt of event data 134 notifying the event processing server 110 of the occurrence of such an event, the event processing server 110 uses stored event information 124-2 to determine how to interpret or handle the event data 134.

Although the initial installation of the system may include an initial installation of event information 130, additional event information 520 may be added to the storage facility 124-2 of event information as it is "discovered". If event information 124 required by the event processing server 110 to process event data 134 has not been provided in advance of receiving the event data 134 (e.g., notifying the event processing server 110 of the occurrence of an event), the event processing server 110 will return an event rejection 136.

An event generation client 210 may also be configured to provide event registration information 130 to the event processing server 110 that serves to notify the event processing server 110, in advance, of event data 134 that it might receive. If the event processing server 110 is missing event information 124 for the event data 134 identified in the event registration information 130 process, the event processing server 110 immediately returns a event rejection 136 of the event registration, notifying the event generation client 110 of the event information 124 that it should provide in order to be able complete event data 134 processing.

One embodiment employs the use of extensible mark-up language (XML) format documents as the communications medium for information exchange between the components of the system. Sources of event data such as application software programs embedded software, and software programs which control hardware may be programmed to provide event data (e.g., including, for example, company, product, module, help, event ID, etc.) using the format provided by the XML templates. In addition, users or customers can generate event information templates of their own in order to accommodate their unique event handling requirements.

Since the event processing server can process text-based XML data, transmitted by means of the commonly available and widespread HTTP protocol, the embodiments of the invention provide a mechanism for event generation clients 210 sending event messages 140, etc. and for event processing servers 110 accepting event information 124 and event data 134, etc. operating on a variety of platforms.

Fig. 4 includes an event processing server 110, two different event generation processes 220-1, 220-2 and representations of an event handling system developer 501, a product developer 502, a customer 502, and technical support personnel 508. As will be explained later, in more detail, the event processing server 110 and the event generation processes 220-1 and 220-2 can access event information 124, 505-3, respectively.

The event processing server 110 shows two event report destination URL's 522, 524. Attached to the event generation processes 220-1, 220-2 are a series of source identifiers 506, 512, 514, 516 used to identify event sources 140-1, 140-2.

The flow of event messages 140, event rejections 136 and report destination address 138 documents depict two different scenarios of operation: i) one scenario that involves the use of event information registration 130 in which the event processing server 110 determines missing event information 124 necessary to process expected future event data 134 during the registration process, and ii) transmission of event data 134-1 in which missing event information 124-5 is accessible to the event processing server 110 without prior registration. Further details of the invention will now be provided with respect to Fig. 5.

Figs. 5-7 are a flow chart of a procedure performed according to the diagram of Fig. 4.

Fig. 5 is a flow chart of an example 600 of the invention as depicted in Fig. 4.

In step 606, event system developers 501 provide templates 504 which include globally unique identifiers (GUID's T1 and T3) for use by product developers 503 when creating products such as event sources 142-1, 142-2. The templates 504 are sample documents demonstrating the proper format (e.g., the proper XML formatting) for event messages 140, etc.

In step 609 the product developer 503 manufacturers products (e.g., events sources 142-1,142-2) containing event information 124. The templates 504 show the format for event message 140 documents (e.g., event registration 130 documents, event

information 124 documents, event data 134-1, 134-2 documents) and other documents that are processed by the event processing server 110.

For example, suppose an interface card manufacturer (e.g., a product developer 503) wants its product to have the ability to output error messages that can be processed by the event processing system 500 according to one embodiment of the invention. In order to incorporate that capability into the product developer's 503 product (e.g., the interface card), it is necessary for the interface card to produce an event detectable by he event generation process which in turn, produces an output which can be transferred to and processed by the event processing server 110. In order to create that capability, the interface card manufacturer (e.g., product developer 503) modifies software drivers which operate the actual interface card hardware (e.g., 142-1, 142-2, etc.) to include functions performed by the event generation client 210, and in particular, includes the ability to create event message 140 documents in the proper format.

In step 612, a customer purchases and installs a product (142-1, 142-2) including an event generation client (210, See Fig. 1, which is capable of operating an event generation process, more specifically 220-1, 220-2, in this example) in the customer facility that operates the event source 142. For example, a customer 502 purchasing an interface card such as the one described above, installs the interface card (e.g., event source 142-1, 142-2, etc.) within his own computer system. As described above, the software drivers provided with the interface card that operate the interface card serve as an event generation client 210.

In step 615, the product (e.g., event source 142-1, 142-2, etc.) begins operation.

In step 618, the event generation client 210 (e.g., operating event generation process 220-1) sends event registration information 130 including identification (e.g., GUID's) of event information 124 required to process event data 134. The event registration 130 informs the event processing server 110 about event data 134-1 that it can expect to receive in the future. Included in the event registration 130, in one section 518, are GUID's (e.g., ABC, DEF, GH representing a company, product, and module,

respectively). The event registration 130 may also include identification of event information 520 (e.g., EI 1, EI 2, EI 4) as described earlier. The event registration information 130 serves to notify the event processing server 110, in advance, of event data 134 that the event generation client 210 is configured to send to the event processing server 110. By virtue of receiving the event registration information 130, the event processing server 110 is able to check the event processing server's 110 existing event information 124-2, in order to determine if all of the existing event information 124-2 is available for processing forth coming event data 134 that will be received. Event registration 130 thereby reduces the likelihood that processing of any event data 134 will be delayed, as a result of not having the proper existing event information 124.

The event generation client 210 may send event registration information 130 to the event processing server 110 at any time. For example, after a customer 502 installs and powers-up new hardware or software (e.g., event sources 140-1, 140-2, etc.) the event generation client 210-1, 210-2 can immediately send event registration information 130 to the event processing server 110 notifying the event processing server 110 of all the additional event data 134 that it can expect to receive. The event generation client 210 (that is capable of operating an event generation process 220-1) can send event registration information 130 to the event processing server 110 at other times as well. For example, if the customer 502 updates the device drivers of the interface card, the new device drivers may have the ability to report the occurrence of additional errors that the prior device drivers were not capable of reporting. Upon installation of the new device drivers, the event generation process 210-1,may, at that time, send additional event registration 130 notifying the event processing server 110 of the additional error messages (e.g., event data 134) that the event processing server 110 can expect to receive in the future.

In step 621, the event processing server 110 receives an event message 140 such as the event registration information 130. The event registration information 130 sent from the event generation client 220-1 includes at least one unique identifier (e.g., a

GUID) identifying the source 142-1, 142-2 of the event data 134-1, 134-2 and providing an identification of event information required to process forth coming event data.

Fig. 6 is a continuation of the procedure described in Fig. 5.

In step 627, the event processing server 110 identifies event information 124 required to process event data 134 based on the event message 140 (e.g., the event processing server 110 identifies the event information 124 required, based on the source 142-1, 142-2 of the event data 134 as indicated in the event registration information 130).

As an example, using event registration information 130, the event processing server 110 may identify the event information 520-1 required to process forth coming event data 134-1.

In step 630, based on event information determined to be required in step 627, the event processing server 110 determines if the existing event information 124-2 (e.g., stored in an information store) is accessible to, or is present in the event processing server 110. For example, in the event registration information 130 sent to the event processing server 110, the event processing server 110 checks to determine if the identified event information 520-1 is accessible by the event processing server 110 (e.g., stored in the existing event information 124-2) that would be needed to process event data for event source 142-1, that is, company ABC, product DEF, module GH.

In step 633, if the existing event information 124-2 is found to not be accessible to the event processing server 110, the event processing server 110 provides an event rejection 136 indicating which missing event information is required. The event rejection 136-1 specifies what additional or missing event information (e.g., EI 4) will be required for processing future event data 134. In other words, the event processing server 110 returns an event rejection 136-1 to the event generation process 220-1 notifying the event generation process 220-1 of the missing event information.

In step 636, the event generation client 210 receives the event rejection 136-1 indicating missing event information 124-5 from an event processing server.

In step 639, the event generation process 220-1 obtains the missing event information from its set of event information 124-3.

In step 642, the event generation client 210 (e.g., operating event generation processes 220-1, 220-2) sends missing event information 124-5 to the event processing server 110.

In step 643 the event processing server 110 receives missing event information 124-5 identified in the event registration message 130, which was sent by the event generation client 210, as described earlier.

Fig. 7 is a continuation of the procedure performed in Figs. 5 and 6.

In step 645, the event processing server 110 provides an event data destination 138 to the event generation process 220-1. The event data destination 138 is used by the event generation client 210-1 as the destination address 522 (e.g., URL 1) that the event processing server 110 uses to receive event data 134. In an alternative embodiment of the invention, a data destination address 524 (e.g., URL 2) is established as a permanent data destination address 524 for reception of all event data 134-2. In the case of such an embodiment, having a fixed destination address, it may not be necessary for the event processing server 110 to send an event data destination (e.g., such as event data destination 138).

In step 648, an event generation process detects an event (e.g., which occurs with respect to a source 142-1).

In step 651, in response to detecting an event, the event generation process 220-1 creates event data 134 (e.g., formats the event data in a markup language format capable of transmission via a hypertext transmission transport protocol).

In step 654, the event generation process 220-1 sends the event data 134-1 to the event processing server 110. The event generation process 220-1 may periodically initiate one or more status checks of sources (e.g., events sources 142-1, 142-2, etc.) in order to produce status check information and forward status check information in the event data 134-1 to the event processing server 110 as confirmation of an operating

communications channel.

In step 657, the event processing server 110 receives the event data 134-1 via the event data destination 138 (e.g., via a URL address 522 specified by the event data destination 138).

In step 660, the event processing server 110 selects the event information 124-2 based on the event data 134-1. In this example the event information 124-2 required to process the event data 134-1 is already accessible to the event processing server 110 due to the fact that such event information 124 could have been provided as a result of event information 124 that was initially stored or which was transmitted to the event processing server 110 along with event registration information 130 or the event information 124 could have been provided to the event processing server 1110 at any time, asdescribed earlier.

In another example, event information 124 may be provided to the event processing server 110 as a result of an event rejection 136 that occurred after event registration 130 or that occurred due to a prior rejection of event data 134 (see Fig. 1).

In step 663, the event processing server 110 generates an event output via the output device 114 from the selected event information 124. As described earlier, the event output may take a variety of forms including output to a computer display, output which is further analyzed and processed, etc., as determined by stored event information 124. In one embodiment, the event processing server 110 reads or accesses first and second event data 134-1, 134-2 and the event processing server 110 then processes the first and second event data 134-1, 134-2, etc. to produce event output data (e.g., displayed via the output device 114) that reflects a hierarchical event relationship between the first and second event data 134-1, 134-2, etc. As described earlier, the event processing server 110 can compare event data 134-1, 134-2, etc. which resulted from different occurrences of events in order to identify relationships between different events that could be used to provided additional information for diagnostic purposes.

The features of the invention may be employed in data communications device and other computerized devices such as those manufactured by Cisco systems, Inc. of San Jose, California.

While this invention has been particularly shown and described with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined by the appended claims.

For example, a data communications network, such as the network 105, described earlier, may include hosts interconnected by various communications devices such as routers, bridges, switches, access servers, gateways, hubs, concentrators, proxy servers, repeaters and so forth which exchange data over an interconnection of data links. Connections may include physical or wireless connections, such devices as modems, transceivers, network interface cards, fiber optic cards, ports, facilities such as T1, fractional-T1, or simple wire connections, etc. that allow the propagation of data between the various devices and hosts.

APPENDIX A:

The following appendix entitled "ET Provider Developer's Guide EDCS-156343" provides a detailed explanation of the software that delivers messages and events using XML documents that follow the protocols for providing distributed cross-Enterprise remote state-based hierarchical alarm reporting and monitoring. This embodiment, however, is to be considered as an example only, and it is to be understood that this example is not meant to be limiting of the invention.

ET Provider Developer's Guide

EDCS-156343

DRAFT

Author: Peter Moss
Date:19 November, 2001
Revision:1

TABLE OF CONTENTS

Definitions

Alarm Object	An object that generally indicates some type of failure condition for some component in a system. Typically, an Alarm Object is created by an Event that signals or raises the Alarm, and has a state that consists of the object being raised (down) or cleared (up), and an Assignment Status that indicates what action support personnel is taking in response to the Alarm.
Application Error	A single-state Alarm Object that is deemed less serious than other types of Alarm Objects. This usually indicates some kind of configuration error. For example, in the ICM product, unmapped dialed numbers or routing script errors are classified as Application Errors. An Application Error generally does not affect the operational status of the system.
CEM	The Cisco Email Manager.
Central Repository Site	Part of an Enterprise Monitoring Site that serves as the receiving point for data from ET Providers and upstream Enterprise Monitoring Sites. This site logs data to a database and distributes data to other intra-Enterprise sites for monitoring and distributes data to an optional downstream Enterprise Monitoring Site.
CIM	Common Information Model. Describes management information and offers a framework for managing system elements across distributed systems.
COM	Microsoft's Component Object Model
CSFS	The Customer Support Forwarding Service. A process used to forward ICM events to client processes.
CMS	Central Monitoring Server. The IIS-5 server that processes XML data from remote systems that are being monitored.
DCOM	Distributed COM
DDSN	Distributed Diagnostics and Service Network.
DTP	The Data Transfer Process. An ICM process used to send event data collected from the CSFS process back to the Customer Support Center.
EMS	Event Management System used to provide event (logging) information for the ICM system.
Enterprise Monitoring Site	An Enterprise that logs and monitors data from ET Providers and other upstream Enterprise Monitoring Sites. The Enterprise Monitoring Site is further subdivided into a Central Repository Site and multiple Satellite Distributor Sites which support client applications.
ET	Synonym for Dynamic State-based Hierarchical Alarm Reporting Using XML and HTTP protocols.
ET Provider	Software that delivers messages and events using XML documents that follow the set of ET protocols described in this document.
GUID	Globally Unique Identifier. A randomly generated unique number. See section 1.6 for more details.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hypertext Transfer Protocol using SSL.
I18N	Internationalization
ICM	Intelligent Contact Manager. Enterprise based call distribution solution.
MDAC	Microsoft Data Access Components
MSMQ	Microsoft Message Queue. A component of Windows NT and Windows 2000 that supports message queueing.
Phone Home	A means of automatically reporting logging events back to the customer support center.
Satellite Distributor Site	

	A site within an Enterprise Monitoring Site that serves client monitoring applications. Each satellite site maintains a database that consists of a subset of the data archived at the Enterprise Central Repository Site.
SNMP	Simple Network Management Protocol. An industry standard mechanism used for collecting information and managing systems. Utilizes the UDP transmission protocol.
Socket	An IPC mechanism that is supported on a variety of platforms. Allows for data to be passed between processes on both local and remote systems.
Syslog	UNIX based logging mechanism similar to the Windows NT Event Viewer.
SSL	Secure Socket Layer. An encrypted transport medium.
TAC	Technical Assistance Center.
TCP	Transmission Control Protocol. A stream-based IP transmission protocol using sockets. The protocol provides reliable delivery of data.
Trap	An implementation for representing an event using the SNMP protocol.
UDP	Unreliable Data Protocol. A message-based IP transmission protocol using sockets. Not guaranteed to be delivered.
UTC	Universal Time Coordinate (aka Greenwich Mean Time (GMT))
UUID	Synonymous with GUID.
XML	Extensible Markup Language.
XSL	Extensible Style-sheet Language.

Introduction

1.1 What Is ET?

ET (Extra Terrestrial, or the follow on to Phone Home) is a framework for providing distributed cross-Enterprise remote state-based hierarchical alarm reporting and monitoring using XML and HTTP protocols. It is unique in a number of ways and provides the following features:

- *Enterprise-scale monitoring solution.* A single solution provider can monitor multiple distributed customer installations. Companies that offer TAC monitoring solutions can offer services to a disparate range of customers in different geographical locations.
- *Remote logging and monitoring of an arbitrary collection of systems.* Event logging systems and SNMP viewers work well in a LAN-based environment but become cumbersome to use in a heterogeneous and distributed environment. The ET system is specifically designed for distributed multi-vendor, multi-customer environments.
- *Reliable delivery of Events.* ET protocols guarantee the delivery of messages sent to the Central Monitoring Server. Mechanisms exist to cache Events in various places if the servers are unavailable or the network is unreachable.

- *Fault tolerant Event delivery and archiving.* The HTTP protocol is used to send data to the server. Redundant servers and databases, and duplex messaging between those servers achieve fault tolerance.
- *24x7 reporting of events and alarm conditions.* A client system sends data back to the Central Monitoring Server in real-time as errors or critical events occur. Customer Support often knows when a system is down before the customer does.
- *Proactive monitoring.* Remote systems that "register" with the server must report to the server (heartbeat) at a set interval or be deemed unreachable.
- *Unified State-based alarm reporting.* All products can use the same reporting system. Alarm states can include Up, Down, Inactive, Impaired, etc. States are tracked as new events associated with the alarm are reported. Additional state information includes what the Customer Support Center is doing about an Alarm. Examples include the acknowledgement of an Alarm, the assignment of an Alarm, the association of the Alarm with a trouble ticket, etc.
- *Scheduled event reporting.* Events can be flagged with different levels of delivery priority. Immediate, End of Hour, End of Day, at a specific future date and time. For example, metrics and statistics can be sent once per day ("midnight run"), etc.
- *Platform independence for remote systems that are being monitored.* The monitoring servers can process events from any platform that can create and deliver XML documents via HTTP. For example, a Linux server could send data to the system via HTTPS using XML and following the ET protocol requirements.
- *Product independence allowing easy integration by other Cisco Business Units or third parties.* The XML Template formats and handshaking protocols will be published by Cisco with user documentation explaining how other products can integrate with the "ET" system. Other companies such as Oracle or HP can create alarm events for their systems in the same format and send them to an "ET" server for monitoring and displaying.
- *Very Secure.* All connections are via HTTPS using SSL 3.x. Standard proxy implementations can be employed using existing firewall technologies. Each connection will be required to authenticate using X.509 certificate technology. Private and public keys will be used for intra and inter server communications. Uses Microsoft IIS-5 server technology with multiple security algorithms (NT authentication, etc.) and ability to filter on specific IP addresses. HTTPS over RAS (Modem-Modem) or HTTPS via VPN also possible for additional security.

Other protocols, such as SNMP, can be securely delivered by tunneling. Note that SNMP v3 supports data encryption.

- *Hierarchical representation of objects and events*. Events are related to a hierarchical object model so they can be displayed in an object-oriented manner with other events. Multiple events can be interrelated to indicate the overall health of the entire system.

- *Dynamic learning*. The server can learn about new products and devices as they send information to the server. As new systems are created, the template architecture of the XML documents allows for new products to be monitored without any advanced knowledge of the product or system. Help files and other documentation can be dynamically requested by the server and sent by the client system.

- *Advanced knowledge of only a single URL is needed.* All remote systems will contact the main URL and request services. The main URL will perform all validation and lookup required and contact the remote system to dynamically inform it of all of the URL addresses it needs as well as the URL of the duplex server.

- *Developed to be sold as a separate product.* Other companies such as HP or Oracle add functionality to their products to leverage the reporting system provided by "ET". These companies could then integrate the functionality into their products and resell the solution.

- *Includes Customer Support Center monitoring and reporting tools.* The ET solution includes a comprehensive monitoring tool (AlarmTracker) that provides real-time display and control of the incoming Event stream. In addition, this tool provides users a mechanism to track the Customer Support Center's response to Events. Reporting tools are also included to perform historical trend analysis of Event data. Database schema is published to allow custom reporting tools to be developed.

- *Integrates with trouble ticket reporting systems.* Interfaces are provided that allows the ET logging database to integrate with third party or custom-built trouble ticket systems.

- *Provides a uniform user interface to view remote log files from different applications.* Currently, PC Anywhere is used as the primary mechanism to view and retrieve log files from the remote customer sites. Not only is this solution slow and undesirable from a security standpoint, but it means that the mechanism to retrieve log files from other products requires a completely different procedure. The ET solution provides a common user interface from

which log files can be retrieved from any supported product. (Note that this feature is not implemented in this phase of the project.).

- *Implements a multi-tiered cooperative support model.* Monitoring servers and monitoring applications can be cascaded in order to synchronously share and process information.
- *Improved scalability.* The current Listener solution does not scale well. The HTTP based system will allow for thousands of connections by leveraging web-farm capabilities of IIS.
- *Rapid development and deployment.* Leverages existing Cisco-CCBU components and utilizes Microsoft web technologies to reduce development effort and time to market.
- *Server-side filtering to provide a subset of archived data to Distributors for its set of clients.* Central Repository Site filtering can be done to limit the amount of data transmitted to and maintained by the Distributor sites.
- *Well-published interfaces to allow third parties and other project teams to create customized client applications and Distributor applications.* The ET project will implement a Version 3.0 AlarmTracker application as part of the end-to-end ET solution, but this does not preclude other development teams from developing their own client applications or their own Distributor applications. The generic Distributor is an HTTP Server application which provides logging and a means to deliver an Event stream to client applications (via HTTP polling). The HTTP protocols will be documented and the Database schema documented to allow development teams to build their own Distributor and client applications.

1.1.1 System Overview

ET is a distributed solution that focuses on a HTTP-based message delivery mechanism (using XML) with HTTP server components at a Central Monitoring Server site to perform logging, monitoring and database archiving functions. The use of documented XML templates provides an open, platform-independent, application-independent solution.

The following figure shows the high-level functional components of the ET architecture.

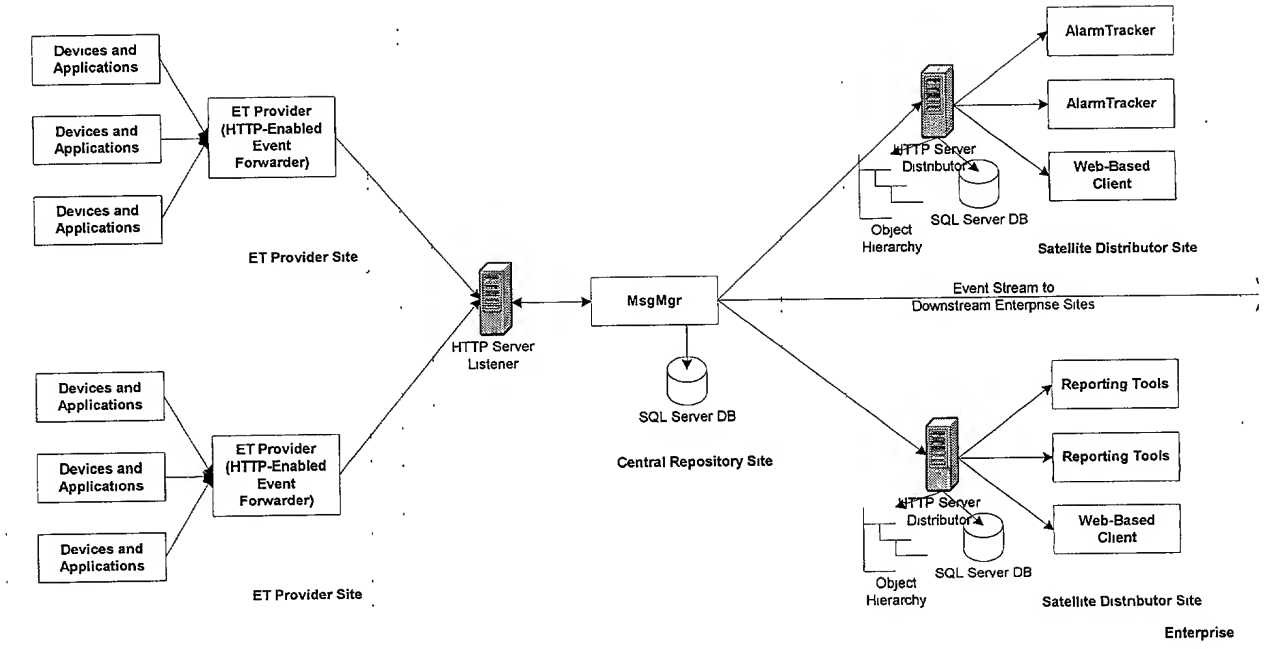

Figure 1 - High-level functional architecture for ET.

The heart of the ET system is the Enterprise Monitoring Site which is defined as an Enterprise organization which monitors incoming data that support the ET protocols. The sources of these data are either ET Providers implemented at customer or Enterprise sites, and upstream Enterprise Monitoring Sites. All data are delivered to an Enterprise via use of the HTTPS protocol. This allows data to be delivered across multiple disconnected Enterprise organizations, and allows these organizations to use the Internet to deliver the data.

The Central Repository Site represents the gateway into an Enterprise Monitoring Site and it also represents a central repository site for archiving current state and historical data in a database.

The HTTP Listener Server represents the gateway into the Enterprise so it is important that the data paths into the Enterprise are secure. To achieve this, the Listener HTTP Servers can be placed outside a corporate firewall or in a DMZ. Data flows in to the MsgMgr component by means of HTTPS POST commands issued by the MsgMgr. Thus, no data are pushed by the Listener through any kind of firewalls that may be in place.

There are no client applications that access the Central Repository Site's Database. This Database is used to log and archive data and to distribute it to Satellite Distributor Sites which serve client applications. This design is chosen so that the Central Repository

Site can focus on its primary task – to receive and log incoming data in a fault-tolerant manner.

Client applications communicate with a Distributor HTTP Server at a Satellite Distributor Site. An arbitrary number of Distributors can exist at an Enterprise Monitoring Site. This distributed design serves two major purposes:

- To distribute the load among multiple servers,
- To move the database closer to client applications to improve network bandwidth utilization and response time.

The ET design also supports the notion of *PassThru* data which allows multiple Enterprises to cooperatively monitor and support applications. By definition, data flows from ET Providers downstream to an Enterprise Monitoring Site. The client applications provide monitoring and support services to monitor and track the state of applications. In addition, all data can *pass through* to downstream Enterprise Monitoring Sites for cooperative or backup monitoring and support. This PassThru model is illustrated in the following figure.

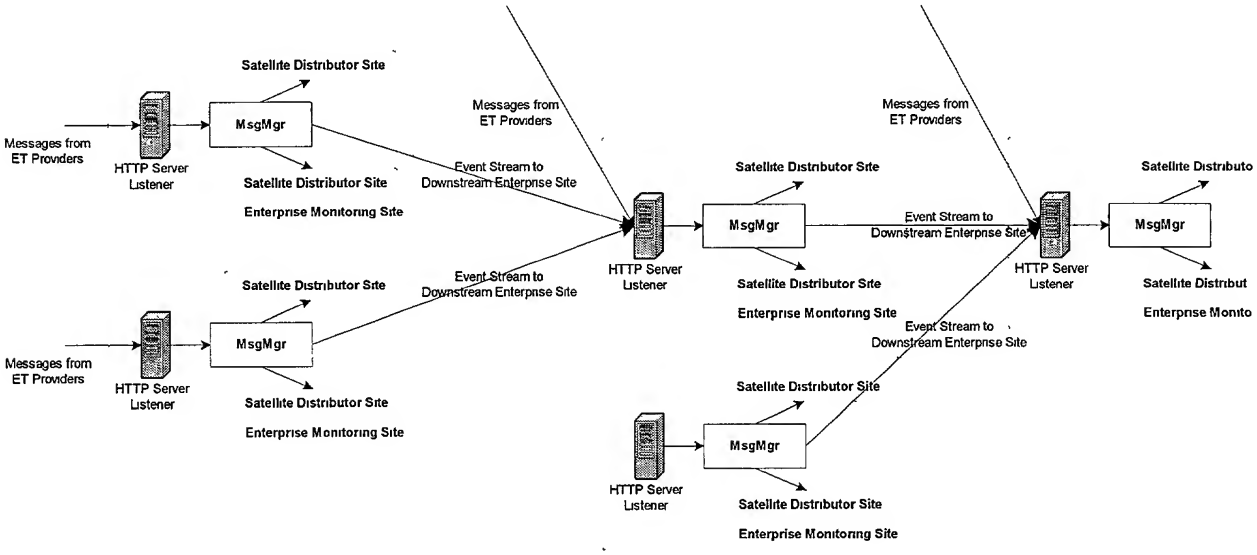

Figure 2 - Generalized PassThru architecture showing data flow from upstream sites on the left toward downstream sites on the right.

The generalized model allows for any Enterprise Monitoring Site to receive data from both an arbitrary number of ET Provider sites and from multiple upstream Enterprise Monitoring Sites.

The ET solution also supports the ability to track what a customer support organization is doing about Alarms that are received at a support site. This is done by associated an additional state variable, an *Assignment Status*, with each Alarm object. When an Alarm occurs, client applications may have the ability to allow users to acknowledge the

Alarm and assign the Alarm to be worked on by someone or to exclude the Alarm from consideration. In some cases, the user may need to manually clear the Alarm. These actions performed by users are referred to as *User Actions* and all User Actions are tracked along with the Alarm object.

User Actions performed by a client application are forwarded by its Distributor for processing and logging by the MsgMgr. In addition, the MsgMgr distributes the User Actions to the other Distributors so that all Distributors see a consistent object state. What is not clear from the above figure is that the User Actions performed at an Enterprise Monitoring Site are also forwarded downstream to any other connected Enterprise Monitoring Site. In addition, if a User Action references an object sent from an upstream Enterprise Monitoring Site, it will also be forwarded to the upstream site's Listener by the MsgMgr. This model provides a basis for a multi-tiered cooperative and integrated customer support system.

1.2 What Is an ET Provider?

An ET Provider is software that tracks object state information and forwards that information to an ET Enterprise Monitoring Site for logging, monitoring and tracking. Object state information includes the definition of *instance nodes,* their *Properties* and *Alarm Attributes* which are special kind of Properties that are driven by Events that indicate the state of these attributes.

One interesting thing about an ET Provider is that there is no spec for how to write one. The authorship of an ET Provider is driven by the format of how the data must be sent from an ET Provider to an Enterprise Monitoring Site. This provides a great deal of flexibility in how an ET Provider is built.

Because the transport protocol is HTTP and XML, it allows an ET Provider to be written on any platform and in any language. Thus, an ET Provider can be written for Linux platforms in Java or on Windows platforms using C++ or C#.

1.3 Responsibility of an ET Provider

An ET Provider author must be able to:

- Describe the systems and applications monitored by means of Class Definitions. Class Definitions can either be created by the author, or existing registered Class Definitions can be reused. The Class Definitions define different classes of objects. A Class Definition includes the definition of its Properties and Alarm Attributes and any Simple Events that may be associated with it.
- Maintain the state of object instances, or *Instance Nodes*. An *Instance Node* is an instantiation of an object that is described by an ET Class Definition. When the state of an instance node changes, the change should be reported to the

Enterprise Monitoring Site for logging, display and tracking. State changes include defined Property values and Alarm Attributes.

- Cache updates to be sent to Enterprise Monitoring Sites in the event that the Listener HTTP Servers are unreachable or down. In this scenario, the updates must be cached and resent when the Listeners are reachable.

All communication with an Enterprise Monitoring Site is done via HTTPS POST commands. The protocols of the message passing between ET Providers and the Listener HTTP Servers at the Enterprise Monitoring Site are described in section 1.12.

It is interesting to note that the ET system is somewhat different than system management systems in that there is (currently) no query mechanism for a central site or client application to *pull* data from the ET Provider site. All data must be *pushed* from the ET Provider to the Enterprise Monitoring Site for logging and display. The reason for this is the focus on *cross-enterprise* monitoring. An Enterprise Monitoring Site support center may be providing logging and monitoring support for a large number of enterprise customers and for a large number of different products and applications. These enterprise customers will each have their own corporate networks and be protected by firewalls. This makes the implementation of bi-directional communication more difficult. We anticipate that a future release of the ET specifications will support some kind of query mechanism by defining new XML templates that can be implemented by ET Providers to allow Enterprise Monitoring Sites to pull data from ET Provider sites. This is not a goal of the first release, however.

1.4 About This Document

This document contains information on how to write an ET Provider and on the XML messaging protocols used to communicate between ET Providers and Enterprise Monitoring Sites.

Section 0 provides a general overview of the ET system.

Section 0 provides a general discussion of the various types of ET elements used in the XML documents.

Section 0 provides specifics on the XML documents that can be sent from the ET Provider to the Listener and the XML responses that the Listener sends back for each of these documents.

Section 0 provides some general guidelines on how an ET Provider communicates with the Listener.

Section 0 describes how I18N is supported within the ET framework.

Section 0 provides a step-by-step checklist to satisfy the requirements of creating an ET Provider.

Section 0 provides a description of the set of Class Definitions that are part of the ET global product namespace.

Appendix A includes a list of the possible Status codes returned in the Listener response XML documents.

Appendix B includes a discussion of Language Identifiers and includes tables that define the set of standard Language Identifiers currently in use.

1.5 Other Documents

ET Functional Specification, EDCS-153155.

ET Elements

The ET specification and protocols are entirely XML based. Thus, all message passing protocols are in the form of XML documents that are transmitted to an HTTP Server via HTTP(S) POST commands. The responses to all commands are also XML documents. This section introduces the types of elements that the ET specification includes. Section 1.12 includes the specific protocols and XML templates that the ET specification defines.

1.6 GUIDs

A Globally Unique Identifier (GUID) (also synonymous with Universally Unique Identifier (UUID)) is not really an ET element per se, but it is a concept that is used throughout the ET protocol to allow ET Providers and client sites to define keys to uniquely identify objects that are guaranteed not to clash with any other provider's or site's keys.

A GUID is a 128-bit value that can be described by the following structure in C:

```
typedef struct _GUID
{
  unsigned long  Data1;
  unsigned short Data2;
  unsigned short Data3;
  unsigned char  Data4[8];
} GUID;
typedef GUID UUID;
```

When represented in text, a GUID is represented like **xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx**, where **x** is a hexadecimal digit (0-f), e.g., 6d5313c0-8c62-11d1-b2cd-006097df8c11. (It is interesting that the C structure shown does not match the text representation exactly.)

Throughout the ET specification, GUIDs are often used to allow ET Providers to specify Product namespaces, Site identifiers and other object identifiers. It is easy to generate a unique GUID that is guaranteed not to clash with anyone else's GUID so the use of a

GUID is a good choice to use as key in many tables. This allows you to select the key and not force the monitoring site to do this for you.

1.7 Product Namespaces

As an ET Provider author, you are likely providing support for a new or existing product, application or collection of hardware. We will refer to these as *products* for the sake of uniformity. If the product has not yet been registered at the Enterprise Monitoring Site, you must register it and provide the appropriate Class Definitions for the instance nodes you are maintaining.

When you register a product with an Enterprise Monitoring Site, you associate a *ProductNamespaceGUID* with it. The GUID is a unique identifier that is guaranteed not to class with other ProductNamespaceGUIDs. The product namespace you define qualifies any Class Definitions you create (see section 1.8). That is, you may define a series of Class Definitions, and in order for their names and IDs not to clash with any other Class Definitions, you use your Product Namespace as a discriminator.

1.7.1 Product Versioning

When you define a set of classes and associate them with a ProductNamespace, you will assign a Version number to those Class Definitions. If your Class Definitions change over time, you must take care to update them in a systematic and controlled fashion. You must be aware that it may be possible for some sites to be reporting data from your Version 1.0 product, some sites to be reporting data from your Version 1.5 product, and others to be reporting data from your Version 2.0 product. Ideally, the Enterprise Monitoring Site software can properly display data from all versions if you manage your Class Definitions properly as your product versions change.

Let's say a site wants to register a Version 2.0 of a product with an Enterprise Monitoring Site. When the site sends a <RegisterProduct> message to the Listener, it will respond with a message that indicates which Version of your product it knows about. If the response comes back with Version='0.0', it means that the product has not been registered with the Enterprise Monitoring Site. If this response is received, the site must send one or more documents that describes all of the product updates that take the product from Version 0.0 (nothing) to Version 2.0. So if you have three known Versions of your product (1.0, 1.5 and 2.0), in this scenario, you must send three <ListenerProductAdmin> XML documents in the following order:

1. A document that describes the transition from Version 0.0 (no definitions) to Version 1.0,
2. A document that describes the transition from Version 1.0 to Version 1.5,
3. A document that describes the transition from Version 1.5 to Version 2.0.

Thus, your ET Provider must be able to send <RegisterProduct> updates for any point release you have created, and they must be sent as "delta documents." That is, each document must describe only the changes of the Class Definitions from one version to the next.

These steps are important in the scenario illustrated in Figure 2 where there are multiple Enterprise Monitoring Sites involved, and they may come on line at different times. The various ET Providers that are feeding these Sites may represent product Versions that range from 1.0 to 2.0. So it is important that each Enterprise Monitoring Site know how to send other downstream Enterprise Monitoring Sites any Version upgrade in the chain.

Thus, your responsibility once you receive the Version number that the Enterprise Monitoring Site knows about is to send a series of documents that will bring the Enterprise Monitoring Site to the latest Version that your ET Provider knows about. If the Listener comes back with Version='1.0', it means that it knows about your Version 1.0 Class Definitions, but nothing newer. In this case, the site must send the set of Class Definitions that represent the "delta" between Version 1.0 and Version 2.0. It is up to the ET Provider to create the appropriate XML document to send that represents the differences. The document must conform to the specifications given in section 1.14 regarding sending Class Definition updates.

If the Listener comes back with Version='2.0', it means that the Enterprise Monitoring Site already knows about this product and is up to date with its Class Definitions. In this case, no Class Definition messages need be sent.

You must be careful about how you modify Class Definitions as your product changes Version. For example, you should probably not delete a Class Definition from one version to another since an older version may still send data for those old class definitions. You should delete a Class Definition only if you know it is never referenced by any instances for any product version. Similarly, you should be very careful about deleting Properties or Alarm Attributes of a class. Generally, it will be safe to add new Class Definitions, and add Properties, Alarm Attributes and Simple Events to existing classes. For more information about updating a product Version and updating Class Definitions, see section 1.14.4.

1.7.2 Product Registration

You register a product with an Enterprise Monitoring Site by sending a special product administration message to the Listener at the Enterprise Monitoring Site. The registration message consists of an XML document with a <ListenerProductAdmin> root tag with a <RegisterProduct> child tag. The <RegisterProduct> contains two important items:

- ***ProductNamespaceGUID*** – this is a GUID that you provide to uniquely identify the product. This GUID is used as a key to identify your product (in case there is

a name clash, for example, if two companies both try to register a product called SuperX). The GUID is used to classify your namespace of Class Definitions.

- ***Name*** – this is a "friendly" name, i.e., the name of your product.

This registration message is described in more detail in section 1.14.

A product is registered in two steps:

1. The site queries the Enterprise Monitoring Site by sending a <ListenerProductAdmin> XML document with a <RegisterProductQuery> message. The response from the Enterprise Monitoring Site contains information on whether that product has been registered before, and if so, what product Version does it currently know about.
2. If the Version returned by the Enterprise Monitoring Site is less than the Version the site currently supports, the ET Provider must then send one or more <ListenerProductAdmin> XML documents, each with a <RegisterProduct> message. The <RegisterProduct> message must contain the set of Class Definitions and updates to Class Definitions to bring the product Version that the Enterprise Monitoring Site knows about up to date. These are the "delta documents" referred to in section 1.7.1. If the Version returned by the Enterprise Monitoring Site is greater than or equal to the Version the site currently knows about, then nothing further needs to be done with respect to product registration.

An example of the messages that are transmitted by the ET Provider to Listener in this sequence is shown in the figure below.

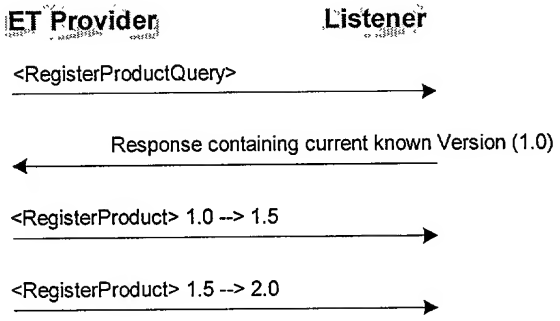

Figure 3 - Message sequence for Product Registration. This example shows a scenario in which the Enterprise Monitoring Site currently knows about Version 1.0 of the product. The ET Provider knows about Version 2.0 so it must send two documents consisting of the 1.0 to 1.5 changes and another one consisting of the 1.5 to 2.0 changes.

The registration messages are described in more detail in section 1.14.

1.8 Class Definitions

1.8.1 Overview

A *Class Definition* describes an object of some type. A class corresponds to the natural idea of a collection of similar things. It is much like a class definition in C++.

When defining a class, it must be part of a Product Namespace. You specify the Product Namespace when you define the class.

When you define a class, you should think about what type of object you are trying to represent. This thought process should be familiar to designers with object-oriented design experience. Defining ET Class Definitions is much like defining SNMP MIBs or CIM Classes. In fact, we have tried to be somewhat compatible with CIM class concepts in the hope that people who currently support CIM may be able to easily write translation software to morph CIM class definitions to ET Class Defintions. However, we are not able to support the depth and breadth of the CIM specification.

A class consists of some basic static definitions such as its name, a unique identifier and a description of what the class describes, and it contains definitions of dynamic variables that each instance of the class possess. The dynamic variables fall into two primary categories:

- ***Properties*** – these are variables that have value and type. Properties are things like number of threads, virtual memory size, number of records, temperature, etc. Properties are described in more detail in section 1.8.4.
- ***Alarm Attributes*** – these are variables that are event driven and represent a binary state of some attribute. The binary state is either Up or Down. Alarm Attributes are described in more detail in section 1.8.5.

Another aspect of a class that does not quite fit into the definition of what a variable is *Stateless Events*. You can associate Stateless Events also known as *Simple Events* with a Class. These Simple Events can also be defined as part of the Class Definition. Simple Events are described in more detail in section 1.8.6.

1.8.2 OIDs

When you define a class, you must specify a unique identifier for it. This is its Object Identifier or OID. A combination of the Product Namespace and the OID must uniquely identify the class. The OID is a 32-bit unsigned integer in the range [1 - 0x7fffffff] that you can select. Note that you must not use any OIDs whose high order bit is set (values from [0x80000000 – 0xffffffff]). This range is reserved. When you register a new class, it is up to you to select a unique value. This is checked against all registered classes so if you fail to pick a unique OID, the class will not be registered. This is something that needs to be managed by the ET Provider.

1.8.3 Inheritance

The ET specification allows for class inheritance. In other words, your class *derives from* some other *base class*. When you define a class, your definition can inherit from another class within your product namespace, or from any class that is part of the ET global namespace.

Multiple inheritance is not supported.

1.8.4 Properties

Properties provide characteristics of a class. Properties are what make class definitions different from each other (along with Alarm Attributes and Simple Events). Properties are like member variables in a C++ class.

Properties are identified by their names. A property name must be unique for that class and for all classes that the class derives from. That is, if class B derives from class A and class A has a property with name Foo, class B cannot have a property of name Foo. This constraint is enforced when the class is registered with the Enterprise Monitoring Site.

In the ET specification, properties must be scalar. Properties are persisted in memory and in databases as a VARIANT data type. A VARIANT data type represents a union of a fairly wide variety of data types. It can be represented in a C structure as follows:

```
typedef struct tagVARIANT  {
  VARTYPE vt;
  unsigned short wReserved1;
  unsigned short wReserved2;
  unsigned short wReserved3;
  union {
    Byte              bVal;                 // VT_UI1.
    Short             iVal;                 // VT_I2.
    long              lVal;                 // VT_I4.
    float             fltVal;               // VT_R4.
    double            dblVal;               // VT_R8.
    VARIANT_BOOL      boolVal;              // VT_BOOL.
    SCODE             scode;                // VT_ERROR.
    CY                cyVal;                // VT_CY.
    DATE              date;                 // VT_DATE.
    BSTR              bstrVal;              // VT_BSTR.
    DECIMAL           FAR* pdecVal          //
VT_BYREF|VT_DECIMAL.
    IUnknown          FAR* punkVal;         // VT_UNKNOWN.
    IDispatch         FAR* pdispVal;        // VT_DISPATCH.
    SAFEARRAY         FAR* parray;          // VT_ARRAY|*.
    Byte              FAR* pbVal;           //
VT_BYREF|VT_UI1.
    short             FAR* piVal;           // VT_BYREF|VT_I2.
    long              FAR* plVal;           // VT_BYREF|VT_I4.
    float             FAR* pfltVal;         // VT_BYREF|VT_R4.
    double            FAR* pdblVal;         // VT_BYREF|VT_R8.
    VARIANT_BOOL      FAR* pboolVal;        //
VT_BYREF|VT_BOOL.
```

```
        SCODE           FAR* pscode;        //
VT_BYREF|VT_ERROR.
        CY              FAR* pcyVal;        // VT_BYREF|VT_CY.
        DATE            FAR* pdate;         //
VT_BYREF|VT_DATE.
        BSTR            FAR* pbstrVal;      //
VT_BYREF|VT_BSTR.
        IUnknown        FAR* FAR* ppunkVal; //
VT_BYREF|VT_UNKNOWN.
        IDispatch       FAR* FAR* ppdispVal; //
VT_BYREF|VT_DISPATCH.
        SAFEARRAY       FAR* FAR* pparray;  // VT_ARRAY|*.
        VARIANT         FAR* pvarVal;       //
VT_BYREF|VT_VARIANT.
        void            FAR* byref;         // Generic ByRef.
        char            cVal;               // VT_I1.
        unsigned short  uiVal;              // VT_UI2.
        unsigned long   ulVal;              // VT_UI4.
        int             intVal;             // VT_INT.
        unsigned int    uintVal;            // VT_UINT.
        char FAR *      pcVal;              // VT_BYREF|VT_I1.
        unsigned short FAR *  puiVal;       //
VT_BYREF|VT_UI2.
        unsigned long FAR *   pulVal;       //
VT_BYREF|VT_UI4.
        int FAR *       pintVal;            //
VT_BYREF|VT_INT.
        unsigned int FAR *    puintVal;
//VT_BYREF|VT_UINT
    };
};
```

You can see from the union above the types of data that the VARIANT structure supports. The ET specification supports a subset of these types as shown the following table.

VARIANT Types Supported by ET Specification			
Types Supported		Types NOT Supported	
C Type	VARTYPE	C Type	VARTYPE
Byte	VT_UI1	DECIMAL*	VT_DECIMAL
Short	VT_I2	IUnknown *	VT_UNKNOWN
long	VT_I4	IDispatch*	VT_DISPATCH
float	VT_R4	SAFEARRAY	VT_ARRAY | *
double	VT_R8	VARIANT	VT_VARIANT
VARIANT_BOOL	VT_BOOL		
SCODE	VT_ERROR		
CY	VT_CY		
DATE	VT_DATE		

BSTR	VT_BSTR		
DECIMAL	VT_DECIMAL		
char	VT_I1		
unsigned short	VT_UI2		
unsigned long	VT_UI4		
int	VT_INT		
unsigned int	VT_UINT		

Table 1 - Table of supported VARIANT types.

The above VARTYPE enums are defined by the following C enum:

```
enum VARENUM {
    VT_EMPTY    = 0,
    VT_NULL     = 1,
    VT_I2       = 2,
    VT_I4       = 3,
    VT_R4       = 4,
    VT_R8       = 5,
    VT_CY       = 6,
    VT_DATE     = 7,
    VT_BSTR     = 8,
    VT_DISPATCH = 9,
    VT_ERROR    = 10,
    VT_BOOL     = 11,
    VT_VARIANT  = 12,
    VT_UNKNOWN  = 13,
    VT_DECIMAL  = 14,
    VT_I1       = 16,
    VT_UI1      = 17,
    VT_UI2      = 18,
    VT_UI4      = 19,
    VT_I8       = 20,
    VT_UI8      = 21,
    VT_INT      = 22,
    VT_UINT     = 23,
    VT_VOID     = 24,
    VT_HRESULT  = 25,
    VT_PTR      = 26,
    VT_SAFEARRAY    = 27,
    VT_CARRAY   = 28,
    VT_USERDEFINED  = 29,
    VT_LPSTR    = 30,
    VT_LPWSTR   = 31,
    VT_RECORD   = 36,
    VT_FILETIME = 64,
    VT_BLOB     = 65,
    VT_STREAM   = 66,
    VT_STORAGE  = 67,
    VT_STREAMED_OBJECT  = 68,
    VT_STORED_OBJECT    = 69,
```

```
    VT_BLOB_OBJECT = 70,
    VT_CF = 71,
    VT_CLSID = 72,
    VT_BSTR_BLOB = 0xfff,
    VT_VECTOR = 0x1000,
    VT_ARRAY = 0x2000,
    VT_BYREF = 0x4000,
    VT_RESERVED = 0x8000,
    VT_ILLEGAL = 0xffff,
    VT_ILLEGALMASKED = 0xfff,
    VT_TYPEMASK = 0xfff
};
```

When you define a Property in a Class Definition, you must supply a VARTYPE value to define its type. These values are integers and must be selected from the set of supported VARTYPEs as shown in Table 1 above.

For any Class Definition you define, you can supply a default value for the Property if you want. Otherwise, it is up to your provider to maintain Property values and to report any object instance Property value changes to the Enterprise Monitoring Site. It is entirely up to you to decide how often you want to communicate Property value changes to the Enterprise Monitoring Site. Note that the Enterprise Monitoring Site will maintain both the value of the Property and the time that the value last changed. This information can be made available to client applications for display.

Note that your systems may not notify you when property values change so it may not be easy for you to determine this. Your internal system may support queries so you could accomplish this by occasionally polling the devices for property value changes and then reporting the changes to the Enterprise Monitoring Site. You must keep in mind that the Enterprise Monitoring Sites can only display what you send to it. Thus, the burden is on you to provide current data wherever possible.

Note that the first implementation of ET does not support Property Set operations. This feature may be supported in future versions of the ET protocol. However, the ET specification does include an optional <Writeable> tag which you can include in your Class Definitions to indicate that the Property is writeable (i.e., that you can perform a Set operation).

More details on the exact syntax for defining Properties is given in section 1.14.

1.8.5 Alarm Attributes

Alarm Attributes really represent a special type of Property. Alarm Attributes are binary properties that represent the state of an attribute of an instance. An Alarm Attribute has one of three states:

- Up
- Down

- Unknown (this is a case where the Enterprise Monitoring Site has no data for the Alarm Attribute)

The state of an Alarm Attribute is driven by Event data from your devices. For example, a Node Manager device can report that the state of some process is down or not running. This is reported via an Event that is handled by your ET Provider. Your provider software must then set the state of the appropriate Alarm Attribute and forward the Event to the Enterprise Monitoring Site for logging and tracking.

When an Event drives the state of an Alarm Attribute to Down, the attribute is said to be in a *raised state*. When an Event drives the state to Up, it is said to be in a *cleared state*.

There are a couple of things that make Alarm Attributes unique:

- Since their states are Event driven, the ET server software can maintain a time history of the series of Events related to the Alarm Attribute. That is, it can track how often the attribute is down and how often it is up. Also, the text that is sent along with the Event is saved and this text may contain meaningful information on why the state of an attribute is Down.

- Along with the state of the Attribute, the ET server software provides a mechanism to track what a support center is doing about the Alarm condition. This is maintained as a separate state variable referred to as the *Assignment Status* of the Alarm. This allows client applications to see not only the state of an Alarm Attribute (Up or Down), but also to see what the support center is doing about the Alarm if it is Down. There are also fields associated with the Alarm that indicate which user performed a user action, who or what queue the Alarm was assigned to, and a Ticket ID field that represents a coupling of the ET server software to an external trouble ticket reporting system.

- When you define Alarm Attributes, you can also define detailed descriptions and actions to be taken when a particular Event is received. In other words, you can provide information that enables an ET client application to see why something has failed, and then view information that may help a support person diagnose and fix the problem.

When you define an Alarm Attribute in a Class Definition, you have the opportunity to define the set of Events that are associated with it. Nominally, there would be two such Events defined: the Raise Event which drives the state of the Alarm Attribute to Down, and the Clear Event which drives the state of the Alarm Attribute to Up. You have the option, however, of defining other Events which may represent different scenarios for how the Alarm Attribute's state transitions to Down or Up. For example, there could be multiple ways for a component to be Down. It could be Down because network

connectivity has been lost, or it could be Down because the database is corrupt. Each of these conditions could be represented as a different Event for the same Alarm Attribute.

To support localization, the ET protocols allow you to define a 32-bit <MsgID> tag that is associated with a <MsgText> tag. The <MsgText> is a text string with substitution arguments represented as %1, %2, etc. When you supply the <MsgText> tag you also include a Language attribute that defines the language identifier1 of the message text. By including multiple <MsgText> tags with different Language attributes, you can have client applications view the Event message text in whatever language is supported. More information on the use of MsgIDs is included in section 1.9 and a further discussion of I18N is given in section 0.

If localization support is not important to you, and you do not need or want to use the substitution argument feature supported by the <MsgText> tag, you can omit the <MsgID> and <MsgText> tags and simply pass the literal message text of the Event when you send the Event at run time.

Typically, Alarm Attributes have two states: Up or Down. The ET specification also supports the notion of *single-state Alarms*. These Alarms are raised by sending a Down Event, but there is no corresponding Up Event that can be associated with them. You can flag this type of Alarm Attribute with a <SingleState> tag when you define the Alarm Attribute.

Alarm Attributes also have a *Severity* associated with them. Some Alarm conditions are more important and severe than others. You can assign a numeric severity ranging from 1 (most severe) to 6 (least severe) to each Alarm Attribute. In addition, a special Alarm type known as an *Application Error* can be defined with a severity of 9. An Application Error is defined as an alarm condition that indicates a problem with an application such as a configuration problem, but not a problem that affects the operation of the device. Application Errors are assumed to be single-state state Alarms. Client applications may choose to display Application Errors in a special way, or allow users to filter them out.

More details on the exact syntax for defining Alarm Attributes is given in section 1.14.

1.8.6 Simple Events

Simple Events are defined as stateless Events that can be associated with an object. They are considered to be informational Events that do not require support personnel to acknowledge them and track their state. They are not associated with attributes, but rather to the object instance itself. They cannot be assigned or tracked by support personnel.

---

1 The language identifier is a standard international numeric abbreviation for a country or geographical region. For example, US English is 1033, Japanese is 1041, etc. A full set of language identifiers can be found in various sources, e.g., MSDN on line by searching for "Language Identifiers."

More details on the exact syntax for defining Simple Events is given in section 1.14.

1.9 MsgIDs

The ET specifications employ the concept of MsgIDs to aid in three specific areas:

1. To minimize the amount of static text that needs to be sent in an Event (Alarm Event and Simple Event),
2. To allow support organizations to customize the text and descriptions they see,
3. To provide multi-language support for client applications.

1.9.1 MsgIDs and Substitution Parameters

The use of MsgIDs is commonly used in Windows applications by providing a message DLL that contains a set of message IDs and message text along with substitution parameters like %1, %2, %3 to indicate the dynamic portion of the message. The ET protocols use MsgIDs in a very similar way.

When you send an Alarm Event or a Simple Event, all you need to do is send a previously defined MsgID along with a set of up to five (5) substitution arguments in your message. Sending this information is in lieu of sending just a raw text string.

For example, for a given product namespace, you can define a message and associate a MsgID which is a 32-bit unsigned integer that must be unique for your product namespace. For each MsgID in your product namespace, you can then define a set of the following attributes to be associated with this MsgID:

- Language Identifier – this is a 16-bit integer value that identifies the language of this set of attributes
- Message Text – this is the message text along with substitution parameters (e.g., %1, %2, ...) in the language specified. The Message Text is something a client application would display in an Event log or Alarm monitoring application (like AlarmTracker).
- Description – this is a more verbose and general description of the meaning and significance of this Event in the language specified. This description might be shown by client application users that want more detail on the meaning of this particular Event.
- Action – this is text in the language specified that describes what action (if any) a support center representative might take to remedy the problem.

Section 1.15 describes the EMS message protocols that allow you to send blocks of these message descriptions to the Enterprise Monitoring Site.

For example, consider the following message description:

MsgID 0xe105007e (3775201406)

Language Identifier 1033

Message Text Peripheral %2 (ID %1) is off-line.

Description The specified peripheral is not visible to the ICR. No call or agent state information is being received by the CallRouter for this site.

Action The scripting algorithms may want to take into account that the peripheral is not on-line. This can be done by using the Peripheral...Online variable and routing calls based on characteristics other than real-time data from the specified peripheral.

If an ET Provider wanted to send this Event, it would only need to send the MsgID and the two substitution arguments %1 and %2 as shown above. The following pseudo-XML shows a sample of what the content of the <ListenerEvent> might look like:

```
<ListenerEvent IID='Router/Router_B/Peripheral
Connections/Wilkes_PG23' OID='34'>
  <AlarmAttribute Name="Seen_By_Router">
    <MsgID Value="3775201406" Time="2001-09-21T18:12:01">
      <Arg>Wilkes_PG23</Arg>
      <Arg>5024</Arg>
    </MsgID>
  </AlarmAttribute>
</ListenerEvent>
```

If a client application were displaying this message it might display like so:

Peripheral Wilkes_PG23 (ID 5204) is off-line.

You can see from the two <Arg> elements included in the <ListenerEvent> how these values are substituted into the general Message Text for this MsgID. Hopefully, you can also see how easy it would be for a client application to display this message in a different language by merely choosing a different Language Identifier for displaying the text.

If your application cannot provide message text in this format, you can send Alarm Events with a MsgID Value='0' and simply include the full message text in the first <Arg> element as follows:

```
<ListenerEvent IID='Router/Router_B/Peripheral
Connections/Wilkes_PG23' OID='34'>
  <AlarmAttribute Name="Seen_By_Router">
    <MsgID Value="0" Time="2001-09-21T18:12:01">
      <Arg>Peripheral Wilkes_PG23 (ID 5204) is off-line.</Arg>
    </MsgID>
  </AlarmAttribute>
</ListenerEvent>
```

The resulting text seen by a client application would be identical to the example using a non-zero MsgID shown above. However, in this scenario, there is no way to provide the message text in any language other than the one the message was sent in. This is a limitation unless you have no need to I18N your product.

1.9.2 Customizing Message Information

The ET model described for supporting MsgIDs allows support organizations (and even individual Distributor sites) to customize the Message Text, Descriptions and Actions for their site. For example, an ET Provider will supply a full set of MsgIDs, Message Texts, Descriptions and Actions for all MsgIDs that it might send. However, the ET Provider can't anticipate how a support organization is supposed to react to problem situations. A support organization might have specific in-house phone numbers that could be called, or a list of names to serve as resources for particular messages, or even a troubleshooting wizard that could be associated with certain MsgIDs. The ET specifications allow Enterprise Monitoring Sites and Distributors to "override" the default ET Provider information with any custom substitutions they like. In this way, each group can have their own highly customized Message Texts, Descriptions and Actions for each MsgID if they wish.

Information on how to supply these overrides is also given in section 1.15

1.9.3 Internationalization

Hopefully, you can see how easy it is to I18N the set of MsgIDs for any given product namespace. By simply supplying sets of MsgIDs with different Language Identifiers, this can be accomplished. The details of how this is done is given in section 1.15 with an additional, more general, discussion of I18N given in section 0.

1.10 Sites

A Site is a source of ET data. A Site is defined in two contexts:

- A Customer Site which represents a customer that is reporting data and events for one or more registered products.
- An Enterprise Monitoring Site which represents a collection point for Customer Sites and other upstream Enterprise Monitoring Sites. Enterprise Monitoring Sites are shown in the dashed lines in Figure 2.

In order for a Site to be monitored by an Enterprise Monitoring Site, it must register with the Enterprise Monitoring Site. It does this by sending an XML document with a <ListenerSiteAdmin> root tag to the Listener.

1.10.1 Site Status

There are actually a number of <ListenerSiteAdmin> messages that can be sent to a Listener:

- <RegisterSite> - registers a Site description to the Enterprise Monitoring Site's Site list. The <RegisterSite> message must be the first site-related message sent. This message defines a new site as a source of ET data.

- <SiteOnline> - indicates that a Site is now online. When a Site is online, it means that it will begin sending data and will adhere to heartbeat reporting requirements specified in the <SiteOnline> message.
- <SiteOffline> - indicates that a Site is now offline. This means that it will cease sending Listener Messages and will no longer adhere to heartbeat reporting requirements.
- <DeleteSite> deletes a Site. This message should be used carefully since it results in the deletion of a Site and all of its associated objects at the Enterprise Monitoring Site. This means that all Instance Node and archived Alarm data will be deleted from all databases.

The details of these messages are described in section 1.15.

When a Site is registered via the <RegisterSite> message, the ET Provider sends the Site Name along with a SiteGUID as a key to uniquely identify it. The inclusion of a GUID is important because a customer name like 'AA' may not be unique. A Site needs to be registered with an Enterprise Monitoring Site only once.

After a Site is registered, it can go online and begin sending instance data to the Listener. A Site goes online by sending a <SiteOnline> message to the Listener. Once a Site is online, it must adhere to requirements about sending heartbeat information to the Listener. This is done so that the Enterprise Monitoring Site can track whether or not the Site is functioning properly. When a Site goes online it sends a <HeartbeatInterval> tag that includes a time interval in minutes that indicates how often it will send heartbeat information to Listener. It is the responsibility of the ET Provider to send a heartbeat message to Listener within the HeartbeatInterval time or MsgMgr will report an Alarm condition that indicates that the Site is not sending Heartbeat messages. This could indicate a problem with the communication link between the Site and the Listener or that there are problems with the Sites message delivery software. This type of Alarm can alert support personnel at the Enterprise Monitoring Site that there is a problem with the Site.

If a Site needs to go offline (because of maintenance, for example), it can send a <SiteOffline> message to the Listener. This message tells the Enterprise Monitoring Site that it is going offline, and that it will not be sending any more heartbeat information to Listener. This way, the Enterprise Monitoring Site will not raise administration Alarms because the Site is not sending heartbeat messages.

Finally, if you really need to delete a Site, a <DeleteSite> message will do this. Deleting a Site results in the deletion of the Site information and all Instance Nodes and data associated with that Site. That includes all historical Alarm information as well.

1.11 Instance Nodes

An Instance Node represents an instance of a Class Definition. The reason we use the word node is that the ET specification requires that each instance node be represented as a node in some kind of object hierarchy. This is described in more detail when we discuss Instance Identifiers below. An Instance Node is like an instance of a C++ class; like saying CFoo* pFoo = new CFoo.

You declare an Instance Node at run time. That is, once you have defined all of you Class Definitions and you have registered a site (see section 1.9), you can begin to declare Instance Nodes that belong to that site. You will mix Instance Node declarations along with Property and Alarm Attribute updates and Simple Events in the Listener Messages stream which are contained in an XML document with a <ListenerMessages> root tag. Instance Node declarations use the <InstanceNodeDeclaration> message type within the <ListenerMessages> XML document.

Each Instance Node must be uniquely identified and classified.

An Instance Node is classified by associating it with a ProductNamespaceGUID and Object Identifier (OID). These two features uniquely identify the type of object the instance node represents and implicitly defines the Properties and Alarm Attributes it includes.

An Instance Node is uniquely identified by a combination of the following three things:

1. A SiteGUID which indicates which site it is part of
2. A ProductNamespaceGUID which indicates which product it is associated with
3. An *Instance Identifier* (IID).

The IID is a text string that places the object in a specific place in an object hierarchy. It is up to the ET Provider to define how the object fits within the object hierarchy. This hierarchy is arbitrary (from the ET server software's point of view), and is used for display purposes in client applications to present a hierarchical order to the set of Instance Nodes monitored. In addition, this hierarchy allows client applications to define rollup logic that may determine how a node's state is represented. For example, the AlarmTracker software represents the overall state of a node as a rollup of the state of all of its Alarm Attributes. That is, a node is shown as red if any of the Alarm Attributes for that node are Down. In addition, the node's state rolls up to its parent node. Other client applications may employ different rollup logic.

The IID is specified by a series of strings separated by the slash '/' character. An example of an IID might be **Network View/Logger A/Managed Processes/rtp**. This IID represents an end node with name **rtp** whose parent node is **Network View/Logger A/Managed Processes** whose parent node is **Network View/Logger A** etc. This syntax

allows for logical groupings of devices and applications that is entirely up to the discretion of the ET Provider to define.

One critical requirement when declaring an Instance Node in the Listener Messages stream is that all of its parent nodes must have been previously declared. In the above example, it is assumed that **Network View/Logger A/Managed Processes, Network View/Logger A** and **Network View** have been previously declared in the Listener Messages stream. If this is not the case, the ET server software has no choice but to create these Instance Nodes as Instance Nodes of a generic class type. This is most likely not what you want. Therefore, it is important that you make sure you have previously declared all parent nodes when declaring a new Instance Node.

1.12 Dynamic Updates

Once an ET Provider has successfully registered the set of products (and the highest Version numbers it knows about) it will be reporting on and once the Site has been registered and put online, it can begin sending dynamic update information that includes the following types of information:

- Instance Node declarations (new Instance Nodes), and Instance Node deletions
- Instance Node Property updates
- Alarm Attribute updates (in the form of Alarm Events)
- Simple Events

This type of information represents the bulk of the dynamic information that the ET Provider is responsible for providing to the Enterprise Monitoring Site.

1.12.1 Instance Node Declarations

One of the most important things for an ET Provider developer to keep in mind is that before reporting any data for an Instance Node, he must send a message to the Listener that describes the object instance. The concept of an object instance or Instance Node was introduced in section 1.11. In order for the ET server software to log and monitor Properties, Alarm Attributes and Simple Events at the Enterprise Monitoring Site, it must know what object is reporting the information. It is up to the ET Provider to send this information in advance of sending the Property update or Event. This is a one-time operation in the sense that once the Instance Node has been declared and identified (via its IID), the ET Provider does not need to send this information again.

Another key thing for an ET Provider to recognize is the need to place the new Instance Node in a pre-existing hierarchy of object instances. The IID implicitly places the Instance Node in this hierarchy by means of its syntax (see section 1.11), but it is up to the ET Provider to make sure that all of the parent nodes are already defined when declaring a new Instance Node.

As an example, let's assume that a site has just come on line and has yet to send any dynamic data. The ET Provider wishes to send information about an Alarm Event that just occurred for a process named rtp which runs on a device called a Logger. The ET Provider has given this process an IID of **Network View/Logger A/Managed Processes/rtp.** Since no dynamic data has yet been sent for any Instance Nodes, the ET Provider must first send Instance Node declarations for each of the rtp process's parent Instance Nodes. In other words, in pseudo code, the ET Provider must send this series of messages to the Enterprise Monitoring Site

```
<DeclareInstanceNode OID='12' IID='Network
View'>...</DeclareInstanceNode>
<DeclareInstanceNode OID='14' IID='Network View/Logger
A'>...</DeclareInstanceNode>
<DeclareInstanceNode OID='17' IID='Network View/Logger A/Managed
Processes'>...</DeclareInstanceNode>
<DeclareInstanceNode OID='25' IID='Network View/Logger A/Managed
Processes/rtp'>...</DeclareInstanceNode>
<ListenerEvent IID='Network View/Logger A/Managed Processes/rtp'
OID='25'>...</ListenerEvent>
```

As you can see, each of the parent Instance Nodes are declared first before the Instance Node for the rtp process is declared. Once the node exists, the actual Listener Event that pertains to the rtp Instance Node can be sent.

If any of the first four <DeclareInstanceNode> messages are omitted, a non-fatal error will result from an unknown Instance Node. By non fatal, we mean that the ET server software needs to create a parent node for you, but does not know what kind of node to create. In this scenario, the ET server software creates a generic Instance Node of type ET_UnknownNodeType which is a class that belongs to the global ET Product Namespace (that is, it can be used by any other product). The ET server software returns an error in the response body of the XML document that is sent that indicates what it did.

So that you don't go through the rest of your life living with this mistake, you have the opportunity to correct this error by issuing a <FixInstanceNode> message to modify the OID of the generic node that the ET server software created. You should do this as soon as possible so that any client display applications will properly display the properties of the Instance Node.

If your software does not do this, and you later send a dynamic update that relates to a Property, Alarm Attribute or Simple Event for one of these generic ET_UnknownNodeType nodes, the ET server software will try to correct your error using the OID that you supply in your <ListenerEvent> message. The only wrinkle here is that the display name of the Instance Node will be left at whatever you sent as part of the IID when the node was implicitly created. This may not always be what you want.

Note that it is up to you, the ET Provider, to maintain the state of all the Instance Nodes that you maintain. When an Instance Node is initially declared, you have the option to

send along all of its current state information (the values of its Properties and states of its Alarm Attributes) (the '...' in the pseudo code above). This is not required, and if not done, the ET server software will supply whatever default values are declared in the Class Definition, or an unknown value if there is no default value.

1.12.2 Property Updates

Property updates are sent to the Enterprise Monitoring Site via a <ListenerEvent> message contained in a <ListenerMessages> XML document. Whenever a device's property changes and you want to send that update to the Enterprise Monitoring Site, you can simply package it in a <ListenerEvent> message. You also want to send the time that the update occurred so that any client application software will have an idea when the value last changed state.

It is up to you to implement filtering so that if your devices send you property updates that do not actually change the value of the property, you do not need to send the update to the Enterprise Monitoring Site. There is no real harm in doing this, except that the time stamp associated with the Property is the one you send. This is interpreted as the time that the Property's value *changed.*

1.12.3 Alarm Events

Alarm Events represent an Event that drives the state of an Alarm Attribute associated with some Instance Node. An ET Provider delivers Alarm Events in much the same way as Property updates are delivered. They are sent as a <ListenerEvent> contained in a <ListenerMessages> XML document sent to the Listener.

Unlike Property values, you may not want to implement filtering on state changes since it is possible that each Event may contain additional information or add to a set of causes for why an Alarm Attribute is in a raised state.

1.12.4 Simple Events

Simple Events represent a stateless Event that is associated with some Instance Node. An ET Provider delivers Simple Events in much the same way as Alarm Attribute updates are delivered. They are sent as a <ListenerEvent> contained in a <ListenerMessages> XML document sent to the Listener.

1.12.5 Heartbeat Messages

Heartbeat messages are messages sent from an ET Provider site to an Enterprise Monitoring Site to indicate that the site is operating normally. The Enterprise Monitoring Site tracks Heartbeat messages and, if they are not received, it will track an Alarm object that indicates the lack of Heartbeat messages. At the Enterprise Monitoring Site these Alarm objects appear in a special administrative Product namespace under the site that is reporting.

When a registered site marks itself as online, it includes information in the message to indicate how often it will send Heartbeat messages to the Listener. The SiteOnline message looks like the following:

```
<p:ListenerSiteAdmin xmlns:p='urn:www.cisco.com:ListenerSiteAdmin'
ETVersion='3.0'>
  <SiteOnline SiteGUID='12345678-1234-1234-1234-1234567890ab'>
    <HeartbeatInterval Source='DTPA' Interval='15' />
    <HeartbeatInterval Source='DTPB' Interval='15' />
    <ReadOnly>false</ReadOnly>
  </SiteOnline>
</p:ListenerSiteAdmin>
```

The above XML document marks the site as being online. It includes two <HeartbeatInterval> elements which indicate that the ET Provider has fault-tolerant processing which means that messages can be sent from one of two sources. The Source attribute in the <HeartbeatInterval> element is used as a discriminator in the Enterprise Monitoring Site to track Heartbeat messages. Once the site comes online, the Listener expects to receive a Heartbeat message from each of these Sources at an interval equal to 15 minutes. It is required that each Source send a Heartbeat message to each of the Listener URLs it knows about. This way, the Enterprise Monitoring Site can track the health of the path from each Source to each Listener. This connection diagram is shown in the figure below:

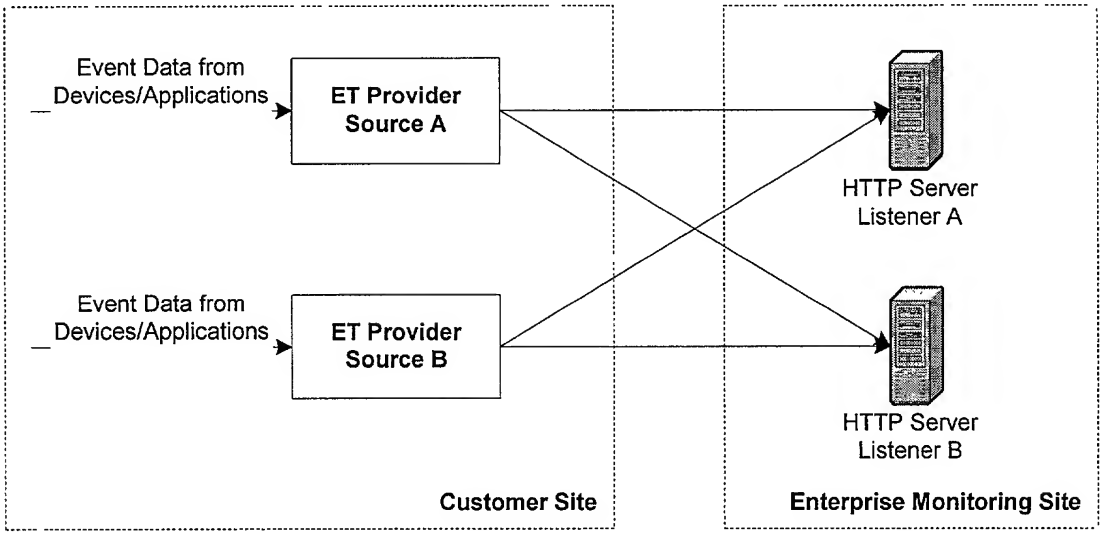

Figure 4 - Paths indicating where Heartbeat messages are sent for a duplexed ET Provider site.

Since the site indicated that it will report from two sources, each of the sources must send Heartbeat messages to each of the two Listeners at an interval specified in the message. If a source fails to send these messages, the Enterprise Monitoring Site will raise an administrative Alarm visible to client applications that indicate the failure to receive Heartbeat messages. Since the Enterprise Monitoring Site knows who should be reporting, the Alarm will indicate the site, the Source, and the particular path to the Listener. This is then tracked as an Alarm object in the ET system.

It is recommended (but not required) that the Heartbeat messages be sent via the same process/mechanism as the other dynamic messages sent to the Listener. The reason for this is that the purpose of the Heartbeat message is to verify that the path from the ET Provider to the Listener is good. If you have a different machine or different process that sends Heartbeat messages, from the ET specifications point of view, you have satisfied the requirements. But if your primary process is wedged or hung or unable to deliver messages, the Enterprise Monitoring Site would never know since it is successfully receiving Heartbeats from an out of band system. By sending the Heartbeat messages in band along with your other messages, if for some reason the messages are not being delivered, at least there will be an Alarm object visible at the Enterprise Monitoring Site that a customer support representative can respond to.

The details of the structure and syntax of Heartbeat messages are found in section 0. The mechanics of how to send Heartbeat messages are described in section 1.22.

ET Protocols

1.13 The Use of XML

XML has emerged as the de facto industry standard for platform-independent data transfer. The combination of using XML, a text-based message protocol, and HTTP make it ideal for communicating information from disparate systems across the Internet. For these reasons, we have chosen to use XML as the basis for all communication between ET Provider sites and Enterprise Monitoring Sites.

In this section we describe the types of communication between the ET Provider sites and the Enterprise Monitoring Site. All communication is initiated by the ET Provider site (this restriction may be relaxed in future releases to allow an Enterprise Monitoring Site to perform queries for information like configuration, Properties and also to allow for Property Set operations) and must be directed to Listener URLs. The Listener represents the gateway into an Enterprise Monitoring Site.

All of the XML message types defined have some kind of XML response written by the Listener. In this section, we will identify the exact syntax of both the message and the response.

1.14 Product Registration and Class Definitions

1.14.1 Overview

Registering a product involves querying the Enterprise Monitoring Site for its current knowledge of the product you want to register, and then supplying whatever Class Definitions it needs based on the response received from the query. This process is described in more detail in section 1.7.2.

Product registration consists of three types of messages that can be included in a <ListenerProductAdmin> XML document that is sent to a Listener:

- <RegisterProductQuery> - this is the first message that should be sent to the Enterprise Monitoring Site. This message queries the Enterprise Monitoring Site for the product Version currently supported.
- <RegisterProduct> - this message is sent to define the set of classes for the product. The content will depend on whether the product is being registered for the first time (the response has a Version='0.0'), or if the Enterprise Monitoring Site contains Class Definitions for an older Version of the product. This message does not have to be sent if the response to the <RegisterProductQuery> contains a Version that is equal to or greater than the Version to be registered.
- <DeleteProduct> - this message is sent to delete an existing product definition. This message has serious implications and can only be sent by an administrator at the Enterprise Monitoring Site.

Once a site determines from the <RegisterProductQuery> message that it must send Class Definitions to the Enterprise Monitoring Site, it does this by sending another <ListenerProductAdmin> XML document, but this time it will contain a <RegisterProduct> message.

The content of the <RegisterProduct> message is a set of <ClassDefinition> messages if a new product is being registered, or a set of <ClassDefinition>, <UpdateClassDefinition> and <DeleteClassDefinition> messages if a product is being updated from an existing Version to a higher Version.

1.14.2 Product Query

To query an Enterprise Monitoring Site for knowledge of a product, an ET Provider sends an XML document to Listener with a root tag of <ListenerProductAdmin>. The XML document must be sent via the HTTP POST method to the following URN:

`https://ListenerURL/ETListener/ETListener.dll?ListenerMessages`

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template2 is used to query an Enterprise Monitoring Site for information about a product:

```
<ListenerProductAdmin ETVersion='3.0'>
    <RegisterProductQuery ProductNamespaceGUID='guid' />
</ListenerProductAdmin>
```

A <ListenerProductAdmin> document must contain exactly one child element, of which <RegisterProductQuery> is one of the allowable types.

The Listener's response to the above document is in the following form:

```
<ListenerProductAdminResponse ETVersion='3.0'>
    <RegisterProductQuery ProductNamespaceGUID='guid' Version='product
version' Status='HRESULT'>Error text if HRESULT is non
zero</RegisterProductQuery>
</ListenerProductAdminResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the message is successfully processed, the Version attribute will contain the product Version that the Enterprise Monitoring Site currently knows about. If the ProductNamespaceGUID is not currently registered (is unknown), the Version returned will be Version='0.0'.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

2 Throughout this document we will refer to XML Templates. In this document an XML Template is something of a cross between an XML schema document and a sample XML document. XML schema documents can often be difficult to read to get a big picture of the contents, and sample documents may not convey enough information. We use XML Templates to try to present a clear picture of the type of XML document that is described.

1.14.3 New Class Definitions

New Class Definitions can be supplied by including a set of <ClassDefinition> tags within a <RegisterProduct> message. These are defined by sending a <ListenerProductAdmin> XML document via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to define new Class Definitions:

```
<ListenerProductAdmin ETVersion='3.0'>
  <RegisterProduct ProductNamespaceGUID='guid' BaselineVersion='old
product version' NewVersion='new product version' Name='product name'
Manufacturer='product manufacturer'>
    <Description>product description [optional]</Description>
      <ClassDefinition OID="Unique 32-bit OID">
        <Name>Classname</Name>
        <BaseClass>Class name from which this class is
derived</BaseClass>
        <Description>Description of object</Description>
        <Properties>
          <Property>
            <Name>Property name</Name>
            <DataType>datatype. We will allow simple VARIANT types
[Note that SQL Server 2000 supports VARIANTS (sql_variant)]</DataType>
            <Description>Description of property</Description>
            <Default>default/initial value</Default>
            <Writeable>true or false</Writeable>  <!-- Writeable
feature not supported in Phase I -->
            <AlarmThresholdHigh>Threshold value. If value is exceeded,
an implicit Alarm object is created.</AlarmThresholdHigh>
            <AlarmThresholdLow>Threshold value. If value drops below
this threshold, an implicit Alarm object is
created.</AlarmThresholdLow>
            <Units>Engineering units</Units>
          </Property>
        </Properties>
        <AlarmAttributes>
          <AlarmAttribute>
            <Name>Attribute name</Name>
            <Description>Description of alarmable
attribute.</Description>
            <Severity>1-6 where 1 is highest severity, or 9 to
indicate an Application Error</Severity>
            <SingleState>true or false</SingleState>
            <MsgIDs>
              <MsgID>32-bit unsigned integer value</MsgID>
              <MsgID>32-bit unsigned integer value</MsgID>
```

```
            <!-- ... set of MsgIDs that pertain to this Alarm
Attribute. -->
            </MsgIDs>
        </AlarmAttribute>
    </AlarmAttributes>

    <SimpleEventMsgIDs>
        <MsgID>32-bit unsigned integer value</MsgID>
        <MsgID>32-bit unsigned integer value</MsgID>
    </SimpleEventMsgIDs>

  </ClassDefinition>

</RegisterProduct>

</ListenerProductAdmin>
```

A <ListenerProductAdmin> document must contain exactly one child element, of which <RegisterProduct> is one of the allowable types.

The <RegisterProduct> element can contain an arbitrary number of <ClassDefinition> tags.

The <RegisterProduct> element contains two Version related attributes:

- BaselineVersion – contains the Version number, e.g., 1.50, that represents the baseline product Version for this update document. For a new product, this value will be 0.0.
- NewVersion – contains the Version number that represents the Version that this document describes.

An error will result if the Enterprise Monitoring Site has no knowledge of the BaselineVersion value (except when BaselineVersion ='0.0'). As described in section 1.7.1, it is up to the ET Provider to provide a complete description in terms of delta documents of the product Version revisions in time and in order.

Note that the <Name> element under the <Property>, <AlarmAttribute> and <SimpleEvent> elements must be unique across all Names for the Class and all of its base classes. That is, if a base class has a Property named Foo, a derived class cannot also have a Property named Foo. This will result in an error in the Class Definition. It is, however, fine to have two different classes which do not share a lineage to both have Properties of name Foo. The same rules apply to <AlarmAttribute>s and <SimpleEvent>s.

The <MsgIDs> and <MsgID> elements deserve some additional comment. To provide support for I18N, the ET protocols allow you to supply generic message text and substitution arguments to make it easier for client applications to view Event data in whatever language is supported. The set of <MsgID> elements in the Class Definition provide a cross-reference to a full set of MsgID values and descriptions for the Product Namespace. The set included in the Class Definition is not required, however. It can be

provided merely as a cross-reference hint to client applications. What is most important, however, is that the full description of the set of <MsgID>s be provided so that if an Instance Node sends an Event with a particular MsgID, it can be decoded properly by client applications. This topic is discussed in more detail in section XXXX.

The Listener's response to a set of <ClassDefinition>s is in the following form:

```
<ListenerProductAdminResponse ETVersion='3.0'>
  <RegisterProduct ProductNamespaceGUID='guid' Status='HRESULT'>
  Error text if HRESULT is non zero
    <ClassDefinition OID='oid' Status='HRESULT'>
      Error text if HRESULT is non zero
    </ClassDefinition>
  </RegisterProduct>
</ListenerProductAdminResponse>
```

The Status attribute associated with the <RegisterProduct> element represents the overall status of processing the document. If the Status value is 0 (S_OK), it means that the whole document was processed without errors. No other child elements will be present in this case.

If the Status value is non-zero, it means that some error occurred in the processing. If an XML error or schema error occurred, there may be no other child elements returned. If there is some kind of logic error related to the processing of individual <ClassDefinition> elements, there will be an associated <ClassDefinition> element in the response document. The processing will abort when the first error is detected. Thus, if more than one error exists in a set of <ClassDefinitions>, only the first error is returned.

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220001 (-2145255423)	MSGMGR_E_KEYALREADYEXISTS	'[OID]' is not unique. It has already been assigned to '[Name]'.
0x80220004 (-2145255422)	MSGMGR_E_NAMEALREADYEXISTS	'[Name]' is not unique. It has already been assigned.
0x00220005 (2228229)	MSGMGR_S_OLDVERSION	The Version number supplied is less than what is currently registered.
0x00220006 (2228230)	MSGMGR_S_DUPLICATEPRODUCTNAME	The product name conflicts with an existing product name. The product name will be further qualified by its manufacturer.
0x80220009 (-2145255417)	MSGMGR_E_UNKNOWNPRODUCTVERSION	The product BaselineVersion is not known.

Note that some of the errors above are actually success codes. A success code in an HRESULT has the high bit (bit 31) cleared. The MSGMGR_S_OLDVERSION and MSGMGR_S_DUPLICATEPRODUCTNAME are examples of success codes.

If a MSGMGR_S_OLDVERSION Status value is returned, it simply means that the Enterprise Monitoring Site has more recent information about a product. In this case, the <RegisterProduct> content is not processed.

If a MSGMGR_S_DUPLICATEPRODUCTNAME Status value is returned, it means that some other site registered a product with the exact same name as you did. This should not happen under normal circumstances, but cannot be entirely prevented. In this case, the ET server software will prepend the name of the product manufacturer to the product name so that it will be registered in the database as ***Manufacturer Name*** instead of just ***Name***. For example, if company **X** registered a product with name **Router**, and later on company **Y** registered a product with name **Router**, a MSGMGR_S_DUPLICATEPRODUCTNAME would be returned and the resulting product name would be registered as **Y Router**.

1.14.4 Updating Class Definitions

When you are updating a set of Class Definitions from one product Version to a higher one, there are two other XML element types that can be used in addition to the <ClassDefinition> element type described in the previous section. When you are updating Class Definitions, the following element types are allowed as children of the <RegisterProduct> message:

- <ClassDefinition> - you are defining a new Class Definition

- <UpdateClassDefinition> - you are updating an existing Class Definition
- <DeleteClassDefinition> - you are deleting an existing Class Definition.

You should be ***extremely*** careful about updating Class Definitions. Your actions may affect a large number of Instance Nodes that match your Class Definition. You should take this into consideration when updating a product Version number.

Generally, there are three types of changes you can make when you update a product Version:

- Addition of new Class Definitions. This is not an issue. These are simply new Class Definitions that your new Version sites may use, but that your older sites will not.
- Updates to existing Class Definitions. These updates come in two types: adding new Properties, Alarm Attributes and Simple Events; and updates or deletions of Properties, Alarm Attributes and Simple Events. Additions are OK except that any of your older object instances will now have uninitialized values for any of the new Properties, etc. that you added. Changes and deletions are problematic. Only cosmetic changes like changing the Description of a Property or Alarm Attribute are reasonable. But if you delete a Property or Alarm Attribute, you will end up in big trouble if you continue to maintain sites that have not upgraded their software. In this situation, any site delivering data from an older version that tries to send an update to a Property or Alarm Attribute that has been deleted will result in an error since that Property no longer exists at the Enterprise Monitoring Site. So you should be very careful about deleting things from one Version to another. You should do so only if you can guarantee that no older version uses this thing (Property, Alarm Attribute, etc.), or if you can upgrade all your site instances together.
- Class Definition deletions. For the same reasons as cited above, this is potentially a very bad thing. You should only delete a Class Definition if you can guarantee that no older version uses this Class Defintion, or if you can upgrade all your site instances together so that no older site will report data for a deleted Class Definition.

When you update an existing Class Definition, you must be aware of the ramifications of such an action.

If you simply add Properties, Alarm Attributes or Simple Events to the definition, this is not a problem. All existing Instance Nodes of this type will pick up these new Properties etc. as part of their definition. For instances from a prior version, these Properties and Alarm Attributes will have uninitialized values.

If you remove Properties, Alarm Attributes or Simple Events from a Class Definition, the associated attributes will be removed from all existing Instance Nodes of that class type.

The following restrictions apply to updating a Class Definition:

- You cannot change the name of a Class Definition or its OID.
- You cannot change the base class of a Class Definition.
- You cannot change the Name or DataType of a Property.
- You cannot change the Name of an Alarm Attribute.

Similarly, if you delete a Class Definition, that will result in the deletion of all Instance Nodes of that type and in the implicit deletion of all Class Definitions (and associated Instance Nodes) from all classes that derive from the deleted Class Definition.

So as you can see, you must take great care when making Class Definition changes from one version to another.

Updating a product Version is done by sending a <ListenerProductAdmin> XML document via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to update a set of Class Definitions:

```
<ListenerProductAdmin ETVersion='3.0'>
    <RegisterProduct ProductNamespaceGUID='guid' BaselineVersion='old
product version' NewVersion='new product version' Name='product name'
Manufacturer='product manufacturer'>
      <Description>product description [optional]</Description>
      <ClassDefinition OID="Unique 32-bit OID">
        <Name>classname</Name>
        <BaseClass>Class name from which this class is
derived</BaseClass>
        <Description>Description of object</Description>
        <Properties>
          <Property>
            <Name>Property name</Name>
            <DataType>datatype. We will allow simple VARIANT types
[Note that SQL Server 2000 supports VARIANTS (sql_variant)]</DataType>
            <Description>Description of property</Description>
            <Default>default/initial value</Default>
            <Writeable>true or false</Writeable>  <!-- Writeable
feature not supported in Phase I -->
            <AlarmThresholdHigh>Threshold value. If value is exceeded,
an implicit Alarm object is created.</AlarmThresholdHigh>
            <AlarmThresholdLow>Threshold value. If value drops below
```

```
this threshold, an implicit Alarm object is
created.</AlarmThresholdLow>
            <Units>Engineering units</Units>
        </Property>
    </Properties>

    <AlarmAttributes>
      <AlarmAttribute>
        <Name>Attribute name</Name>
        <Description>Description of alarmable
attribute.</Description>
        <Severity>1-6 where 1 is highest severity, or 9 to
indicate an Application Error</Severity>
        <SingleState>true or false</SingleState>
        <MsgIDs>
          <MsgID>32-bit unsigned integer value</MsgID>
          <MsgID>32-bit unsigned integer value</MsgID>
          <!-- ... set of MsgIDs that pertain to this Alarm
Attribute. -->
        </MsgIDs>
      </AlarmAttribute>
    </AlarmAttributes>

    <SimpleEventMsgIDs>
      <MsgID>32-bit unsigned integer value</MsgID>
      <MsgID>32-bit unsigned integer value</MsgID>
    </SimpleEventMsgIDs>

  </ClassDefinition>

  <UpdateClassDefinition OID="Unique 32-bit OID">
    <Description>Description of object</Description>
    <Properties>
      <AddProperty>
        <Name>Property name</Name>
        <DataType>datatype. We will allow simple VARIAND types
[Note that SQL Server 2000 supports VARIANTS (sql_variant)]</DataType>
        <Description>Description of property</Description>
        <Default>default/initial value</Default>
        <Writeable>True or False</Writeable>  <!-- Writeable
feature not supported in Phase 1 -->
        <AlarmThresholdHigh>Threshold value. If value is exceeded,
an implicit Alarm object is created.</AlarmThresholdHigh>
        <AlarmThresholdLow>Threshold value. If value drops below
this threshold, an implicit Alarm object is
created.</AlarmThresholdLow>
        <Units>Engineering units</Units>
      </AddProperty>
      <UpdateProperty Name='name'>
        <Description>Description of property</Description>
        <Default>default/initial value</Default>
        <Writeable>True or False</Writeable>  <!-- Writeable
feature not supported in Phase 1 -->
        <AlarmThresholdHigh>Threshold value. If value is exceeded,
an implicit Alarm object is created.</AlarmThresholdHigh>
```

```
                <AlarmThresholdLow>Threshold value.  If value drops below
this threshold, an implicit Alarm object is
created.</AlarmThresholdLow>
                <Units>Engineering units</Units>
            </UpdateProperty>
            <DeleteProperty Name='name' />
        </Properties>

        <AlarmAttributes>
            <AddAlarmAttribute>
                <Name>Attribute name</Name>
                <Description>Description of alarmable
attribute.</Description>
                <Severity>1-6 where 1 is highest severity, or 9 to
indicate an Application Error</Severity>
                <SingleState>true or false</SingleState>
                <MsgIDs>
                    <MsgID>32-bit unsigned integer value</MsgID>
                    <MsgID>32-bit unsigned integer value</MsgID>
                    <!-- ... set of MsgIDs that pertain to this Alarm
Attribute. -->
                </MsgIDs>
            </AddAlarmAttribute>

            <UpdateAlarmAttribute Name='name'>
                <Description>Description of alarmable
attribute.</Description>
                <Severity>1-6 where 1 is highest severity, or 9 to
indicate an Application Error</Severity>
                <SingleState>True or False. Default is
False.</SingleState>
                <MsgIDs>
                    <MsgID>32-bit unsigned integer value</MsgID>
                    <MsgID>32-bit unsigned integer value</MsgID>
                    <RemoveMsgID>32-bit unsigned integer value</RemoveMsgID>
                    <!-- ... set of MsgIDs that pertain to this Alarm
Attribute. -->
                </MsgIDs>
            </UpdateAlarmAttribute>

            <DeleteAlarmAttribute Name='name' />

        </AlarmAttributes>

        <SimpleEventMsgIDs>
            <MsgID>32-bit unsigned integer value</MsgID>
            <MsgID>32-bit unsigned integer value</MsgID>
            <RemoveMsgID>32-bit unsigned integer value</RemoveMsgID>
        </SimpleEventMsgIDs>

    </UpdateClassDefinition>

    <DeleteClassDefinition OID="32-bit OID" />

</RegisterProduct>
```

```
</ListenerProductAdmin>
```

The <RegisterProduct> tag can contain an arbitrary number of <ClassDefinition>, <UpdateClassDefinition> and <DeleteClassDefinition> tags.

The <RegisterProduct> element contains two Version-related attributes:

- BaselineVersion – contains the Version number, e.g., 1.50, that represents the baseline product Version for this update document. For a new product, this value will be 0.0.
- NewVersion – contains the Version number that represents the Version that this document describes.

An error will result if the Enterprise Monitoring Site has no knowledge of the BaselineVersion value (except when BaselineVersion='0.0'). As described in section 1.7.1, it is up to the ET Provider to provide a complete description in terms of delta documents of the product Version revisions in time and in order..

The <MsgIDs> and <MsgID> elements deserve some additional comment. To provide support for I18N, the ET protocols allow you to supply generic message text and substitution arguments to make it easier for client applications to view Event data in whatever language is supported. The set of <MsgID> elements in the Class Definition provide a cross-reference to a full set of MsgID values and descriptions for the Product Namespace. The set included in the Class Definition is not required, however. It can be provided merely as a cross-reference hint to client applications. What is most important, however, is that the full description of the set of <MsgID>s be provided so that if an Instance Node sends an Event with a particular MsgID, it can be decoded properly by client applications. This topic is discussed in more detail in section XXXX.

In the case of an <UpdateAlarmAttribute> or <UpdateSimpleEvent> element, the set of <MsgID>s contained in the <MsgIDs> element will do a full replacement of what was there for the previous Version.

The Listener's response to the <RegisterProduct> document is identical to the one described in section 1.14.3.

In addition to the errors shown in section 1.14.3, these additional errors may be returned:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220001 (-2145255423)	MSGMGR_E_KEYALREADYEXISTS	'[OID]' is not unique. It has already been assigned to '[Name]'.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[OID]|[Name]' does not exist.
0x80220004 (-2145255422)	MSGMGR_E_NAMEALREADYEXISTS	'[Name]' is not unique. It has already been assigned.
0x00220005 (2228229)	MSGMGR_S_OLDVERSION	The Version number supplied is less than what is currently registered.
0x00220006 (2228230)	MSGMGR_S_DUPLICATEPRODUCTNAME	The product name conflicts with an existing product name. The product name will be further qualified by its manufacturer.
0x80220009 (-2145255417)	MSGMGR_E_UNKNOWNPRODUCTVERSION	The product BaselineVersion is not known.

Note that some of the errors above are actually success codes. A success code in an HRESULT has the high bit (bit 31) cleared. The MSGMGR_S_OLDVERSION and MSGMGR_S_DUPLICATEPRODUCTNAME are examples of success codes.

If a MSGMGR_S_OLDVERSION Status value is returned, it simply means that the Enterprise Monitoring Site has more recent information about a product. In this case, the <RegisterProduct> content is not processed.

If a MSGMGR_S_DUPLICATEPRODUCTNAME Status value is returned, it means that some other site registered a product with the exact same name as you did. This should not happen under normal circumstances, but cannot be entirely prevented. In this case, the ET server software will prepend the name of the product manufacturer to the product name so that it will be registered in the database as ***Manufacturer Name*** instead of just ***Name***. For example, if company **X** registered a product with name **Router**, and later on company **Y** registered a product with name **Router**, a MSGMGR_S_DUPLICATEPRODUCTNAME would be returned and the resulting product name would be registered as **Y Router**.

1.14.5 Product Deletion

To delete a product, the sender must be a member of the ETAdmin user group. To delete a product, an administrator (presumably at an Enterprise Monitoring Site) sends an XML document to Listener with a root tag of <ListenerProductAdmin>. The XML document must be sent via the HTTP POST method to the following URN:

https://ListenerURL/ETListenerInt/ETListener.dll?ListenerAdmin

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template3 is used to delete a product.

```
<ListenerProductAdmin Version='3.0'>
    <DeleteProduct ProductNamespaceGUID='guid' />
    <!-- The above message deletes a Product. -->
    <!-- This message can only be processed by a user in the ETAdmin
group -->
</ListenerProductAdmin>
```

A <ListenerProductAdmin> document must contain exactly one child element, of which <DeleteProduct> is one of the allowable types.

Note that the ETListenerInt Virtual Directory is has different access rights than the ETListener Virtual Directory. In particular, only users who are members of the ETAdmin group has access to this Virtual Directory. Thus, anyone not a member of the ETAdmin group who sends a DeleteProduct message to this directory will get an HTTP 401 – Unauthorized HTTP response.

The Listener's response to a ListenerProductAdmin document is in the following form:

```
<ListenerProductAdminResponse Version='3.0'>
    <DeleteProduct ProductNamespaceGUID='guid' Status='HRESULT'>Error
text if HRESULT is non zero</DeleteProduct>
</ListenerProductAdminResponse>
```

For each <DeleteProduct> tag sent to Listener, Listener will include a corresponding <DeleteProduct> tag in the response document with matching ProductNamspaceGUID.

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-	ET_E_INVALIDSCHEMA	Invalid schema.

3 Throughout this document we will refer to XML Templates. In this document an XML Template is something of a cross between an XML schema document and a sample XML document. XML schema documents can often be difficult to read to get a big picture of the contents, and sample documents may not convey enough information. We use XML Templates to try to present a clear picture of the type of XML document that is described.

2145386495)		
0x80200002 (-2145386494)	ET_E_INSUFFICIENTPRIVILEGE	Insufficient privileges to execute this command.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	ProductNamespaceGUID does not exist.

If an ET Provider sends a DeleteProduct message to the ETListener Virtual Directory, the message will be rejected by MsgMgr when it is processed. In this case, the Status sent back will be ET_E_INSUFFICIENTPRIVILEGE.

1.14.6 XML Schemas

The <ListenerProductAdmin> XML document adheres to the following XML Schema document:

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="urn:www.cisco.com:ListenerProductAdminSchema"
targetNamespace="urn:www.cisco.com:ListenerProductAdminSchema">
<xsd:element name="ListenerProductAdmin">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element name='RegisterProductQuery'>
          <xsd:complexType>
            <xsd:attribute name="ProductNamespaceGUID" type="tns:GUID"
use="required" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name='DeleteProduct'>
          <xsd:complexType>
            <xsd:attribute name="ProductNamespaceGUID" type="tns:GUID"
use="required" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name='RegisterProduct'>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name='Description' type='xsd:string'
minOccurs='0' />
              <xsd:element name='ClassDefinition' minOccurs='0'
maxOccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name='Name' type='xsd:string' />
                    <xsd:element name='BaseClass' type='xsd:string'
minOccurs='0' />
                    <xsd:element name='Description' type='xsd:string'
minOccurs='0' />
                    <xsd:element name='Properties' minOccurs='0'>
                      <xsd:complexType>
                        <xsd:sequence>
```

```
                    <xsd:element name='Property' minOccurs='0'
maxOccurs='unbounded'>
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name='Name'
type='xsd:string' />
                                <xsd:element name='DataType'
type='xsd:unsignedShort' />
                                <xsd:element
name='Description' type='xsd:string' minOccurs='0' />
                                <xsd:element name='Default'
type='xsd:string' minOccurs='0' />
                                <xsd:element
name='Writeable' type='xsd:boolean' minOccurs='0' default='false' />
                                <xsd:element
name='AlarmThresholdHigh' type='xsd:string' minOccurs='0' />
                                <xsd:element
name='AlarmThresholdLow' type='xsd:string' minOccurs='0' />
                                <xsd:element name='Units'
type='xsd:string' minOccurs='0' />
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name='AlarmAttributes'
minOccurs='0'>
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name='AlarmAttribute'
minOccurs='0' maxOccurs='unbounded'>
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name='Name'
type='xsd:string' />
                                <xsd:element
name='Description' type='xsd:string' minOccurs='0' />
                                <xsd:element name='Severity'
type='tns:ETSeverity' />
                                <xsd:element
name='SingleState' type='xsd:boolean' />
                                <xsd:element name='MsgIDs'
minOccurs='0'>
                                    <xsd:complexType>
                                        <xsd:sequence>
    <xsd:element name='MsgID' type='xsd:unsignedInt' minOccurs='0'
maxOccurs='unbounded' />
                                        </xsd:sequence>
                                    </xsd:complexType>
                                </xsd:element>
                            </xsd:sequence>
                        </xsd:complexType>
```

```
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name='SimpleEventMsgIDs'
minOccurs='0'>
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name='MsgID'
type='xsd:unsignedInt' minOccurs='0' maxOccurs='unbounded' />
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="OID" type="xsd:unsignedInt"
use="required" />
</xsd:complexType>
</xsd:element>
<xsd:element name='UpdateClassDefinition' minOccurs='0'
maxOccurs='unbounded'>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name='Description' type='xsd:string'
minOccurs='0' />
            <xsd:element name='Properties' minOccurs='0'>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name='AddProperty'
minOccurs='0' maxOccurs='unbounded'>
                            <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element name='Name'
type='xsd:string' />
                                    <xsd:element name='DataType'
type='xsd:unsignedShort' />
                                    <xsd:element
name='Description' type='xsd:string' minOccurs='0' />
                                    <xsd:element name='Default'
type='xsd:string' minOccurs='0' />
                                    <xsd:element
name='Writeable' type='xsd:boolean' minOccurs='0' default='false' />
                                    <xsd:element
name='AlarmThresholdHigh' type='xsd:string' minOccurs='0' />
                                    <xsd:element
name='AlarmThresholdLow' type='xsd:string' minOccurs='0' />
                                    <xsd:element name='Units'
type='xsd:string' minOccurs='0' />
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                        <xsd:element name='UpdateProperty'
minOccurs='0' maxOccurs='unbounded'>
                            <xsd:complexType>
                                <xsd:sequence>
```

```
                                        <xsd:element
name='Description' type='xsd:string' minOccurs='0' />
                                        <xsd:element name='Default'
type='xsd:string' minOccurs='0' />
                                        <xsd:element
name='Writeable' type='xsd:boolean' minOccurs='0' default='false' />
                                        <xsd:element
name='AlarmThresholdHigh' type='xsd:string' minOccurs='0' />
                                        <xsd:element
name='AlarmThresholdLow' type='xsd:string' minOccurs='0' />
                                        <xsd:element name='Units'
type='xsd:string' minOccurs='0' />
                                    </xsd:sequence>
                                    <xsd:attribute name="Name"
type="xsd:string" use="required" />
                                </xsd:complexType>
                            </xsd:element>
                            <xsd:element name='DeleteProperty'
minOccurs='0' maxOccurs='unbounded'>
                                <xsd:complexType>
                                    <xsd:attribute name="Name"
type="xsd:string" use="required" />
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name='AlarmAttributes'
minOccurs='0'>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name='AddAlarmAttribute'
minOccurs='0' maxOccurs='unbounded'>
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name='Name'
type='xsd:string' />
                                        <xsd:element
name='Description' type='xsd:string' minOccurs='0' />
                                        <xsd:element name='Severity'
type='tns:ETSeverity' />
                                        <xsd:element
name='SingleState' type='xsd:boolean' />
                                        <xsd:element name='MsgIDs'
minOccurs='0'>
                                            <xsd:complexType>
                                                <xsd:sequence>
                                                    <xsd:element name='MsgID' type='xsd:unsignedInt' minOccurs='0'
maxOccurs='unbounded' />
                                                </xsd:sequence>
                                            </xsd:complexType>
                                        </xsd:element>
                                    </xsd:sequence>
```

```
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name='UpdateAlarmAttribute'
minOccurs='0' maxOccurs='unbounded'>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element
name='Description' type='xsd:string' minOccurs='0' />
                            <xsd:element name='Severity'
type='tns:ETSeverity' />
                            <xsd:element
name='SingleState' type='xsd:boolean' />
                            <xsd:element name='MsgIDs'
minOccurs='0'>
                                <xsd:complexType>
                                    <xsd:sequence>
  <xsd:element name='AddMsgID' type='xsd:unsignedInt' minOccurs='0'
maxOccurs='unbounded' />
  <xsd:element name='RemoveMsgID' type='xsd:unsignedInt' minOccurs='0'
maxOccurs='unbounded' />
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name="Name"
type="xsd:string" use="required" />
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name='DeleteAlarmAttribute'
minOccurs='0' maxOccurs='unbounded'>
                    <xsd:complexType>
                        <xsd:attribute name="Name"
type="xsd:string" use="required" />
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name='SimpleEventMsgIDs'
minOccurs='0'>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name='AddMsgID'
type='xsd:unsignedInt' minOccurs='0' maxOccurs='unbounded' />
                <xsd:element name='RemoveMsgID'
type='xsd:unsignedInt' minOccurs='0' maxOccurs='unbounded' />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:sequence>
<xsd:attribute name="OID" type="xsd:unsignedInt"
use="required" />
```

```
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name='DeleteClassDefinition' minOccurs='0'
maxOccurs='unbounded'>
                    <xsd:complexType>
                        <xsd:attribute name="OID" type="xsd:unsignedInt"
use="required" />
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="ProductNamespaceGUID" type="tns:GUID"
use="required" />
            <xsd:attribute name="BaselineVersion" type="xsd:float"
use="required" />
            <xsd:attribute name="NewVersion" type="xsd:float"
use="required" />
            <xsd:attribute name="Name" type="xsd:string"
use="required" />
            <xsd:attribute name="Manufacturer" type="xsd:string"
use="required" />
        </xsd:complexType>
    </xsd:element>
  </xsd:choice>
  <xsd:attribute name="ETVersion" type="xsd:float" use="required" />
 </xsd:complexType>
</xsd:element>

<xsd:simpleType name='GUID'>
 <xsd:restriction base='xsd:string'>
  <xsd:pattern value="[0-9A-Fa-f]{8}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{4}-
[0-9A-Fa-f]{4}-[0-9A-Fa-f]{12}" />
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name='ETSeverity'>
 <xsd:restriction base='xsd:unsignedByte'>
  <xsd:minInclusive value='1' />
  <xsd:maxInclusive value='9' />
 </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

This document is delivered as part of the installation kit as **ListenerProductAdmin.xsd**. It can also be retrieved from the Listener via a query (see section XXXX).

1.14.7 XML Tag Descriptions

The following XML tags are defined in the <ListenerProductAdmin> document that are relevant to Product registration:

Tag: **<ListenerProductAdmin>** (root element)

Description: Defines the root element for the ListenerProductAdmin document. This type of document contains product registration and/or Class Definition elements only.
Content Type: elementOnly
Data Type: N/A
Child Elements: <RegisterProductQuery> [0,1][4]
<RegisterProduct> [0,1]
<DeleteProduct> [0,1]

Attribute: ETVersion
Description: ET protocol version number of document. Must be 3.0.
DataType: float
Use: required
Value: 3.0

Tag: <RegisterProductQuery>
Description: Queries an Enterprise Monitoring Site for information about a product.
Content Type: none
Data Type: N/A
Child Elements: none

Attribute: ProductNamespaceGUID
Description: GUID that uniquely identifies the product.
DataType: uuid
Use: required

Tag: <RegisterProduct>
Description: Registers a product with the Enterprise Monitoring Site.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Description> [0,1]
<ClassDefinition> [0,…]
<UpdateClassDefinition> [0,…]
<DeleteClassDefinition> [0,…]

Attribute: ProductNamespaceGUID
Description: GUID that uniquely identifies this product.
DataType: uuid
Use: required

Attribute: BaselineVersion
Description: Baseline product Version number. This is the product Version that this document is updating from. For new product declarations, this value must be 0.0.
DataType: float

---

4 The notation in brackets next to the Child Elements tags represent the minOccurs and maxOccurs values for the tag. A value of … is equivalent to maxOccurs='unbounded'. For example, [0,…] is minOccurs='0' maxOccurs='unbounded'. A value of [1,1] is equivalent to minOccurs='1' maxOccurs='1' and means that the tag must appear and can only appear once.

Use: required

Attribute: NewVersion
Description: New product Version number. This is the Version that this document is updating to.
DataType: float
Use: required

Attribute: Name
Description: Friendly name for product. This name will be used in any client displays.
DataType: string
Use: required

Attribute: Manufacturer
Description: Product manufacturer.
DataType: string
Use: required

Tag: <DeleteProduct>
Description: Deletes a product previously registered with the Enterprise Monitoring Site.
Content Type: empty
Data Type: N/A
Child Elements: none

Attribute: ProductNamespaceGUID
Description: GUID that uniquely identifies this product.
DataType: uuid
Use: required

Tag: <Description>
Description: English (Language Identifier = 1033) text that describes what the product/Class Definition/Property/Alarm Attribute/Simple Event is.
Content Type: textOnly
Data Type: string (max of 256 characters)
Child Elements: none

Tag: <ClassDefinition>
Description: Registers a new Class Definition with the Enterprise Monitoring Site.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Name> [1,1]
<DerivedFrom> [0,1]
<Description> [0,1]
<Properties> [0,1]
<AlarmAttributes> [0,1]
<SimpleEventMsgIDs> [0,1]

Attribute: OID

Description: Unique (within a product namespace) Object Identifier that acts as a key to this class. (Note the combination of the ProductNamespaceGUID and OID must be unique across all product namespaces.) This value can be in the range [1 – 0x7fffffff].
DataType: unsignedInt
Use: required

Tag: <UpdateClassDefinition>
Description: Updates an existing Class Definition.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Description> [0,1]
<Properties> [0,1]
<AlarmAttributes> [0,1]
<SimpleEventMsgIDs> [0,1]

Attribute: OID
Description: Object Identifier of Class Definition to update.
DataType: unsignedInt
Use: required

Tag: <DeleteClassDefinition>
Description: Deletes a previously defined Class Definition.
Content Type: empty
Data Type: N/A
Child Elements: none

Attribute: OID
Description: Object Identifier.
DataType: unsignedInt
Use: required

Tag: <Name> (child of <ClassDefinition>, <UpdateClassDefinition>)
Description: Name of the class. This name must be unique within a product namespace.
Content Type: textOnly
Data Type: string (max of 128 characters)
Child Elements: none

Tag: <BaseClass>
Description: Base class for defined class. If present, the base class must have been previously defined. The BaseClass must be part of the current product namespace, or one of the classes defined in the global ET product namespace (see section 0.
Content Type: textOnly
Data Type: string (max of 128 characters)
Child Elements: none

Tag: <Properties>
Description: Collection node that contains <Property> definitions.

Content Type: elementOnly
Data Type: N/A
Child Elements: <Property> (for <ClassDefinition>) [0,…]
<AddProperty> (for <UpdateClassDefinition>) [0,…]
<UpdateProperty> (for <UpdateClassDefinition>) [0,…]
<DeleteProperty> (for <UpdateClassDefinition>) [0,…]

Tag: <AlarmAttributes>
Description: Collection node that contains <AlarmAttribute> definitions.
Content Type: elementOnly
Data Type: N/A
Child Elements: <AlarmAttribute> (for <ClassDefinition>) [0,…]
<AddAlarmAttribute> (for <UpdateClassDefinition>) [0,…]
<UpdateAlarmAttribute> (for <UpdateClassDefinition>) [0,…]
<DeleteAlarmAttribute> (for <UpdateClassDefinition>) [0,…]

Tag: <SimpleEventMsgIDs>
Description: Collection node that contains a set of MsgIDs representing Simple Events to be associated with this class. This element is optional and the set of MsgIDs is purely ancillary information
Content Type: elementOnly
Data Type: N/A
Child Elements: <MsgID> (for <ClassDefinition>) [0,…]
<AddMsgID> (for <UpdateClassDefinition>) [0,…]
<RemoveMsgID> (for <UpdateClassDefinition>) [0,…]

Tag: <Property> (for <ClassDefintion>)
<AddProperty> (for <UpdateClassDefinition>)
Description: Defines a new Property for a Class Definition.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Name> [1,1]
<DataType> [1,1]
<Description> [0,1]
<Default> [0,1]
<Writeable> [0,1]
<AlarmThresholdHigh> [0,1]
<AlarmThresholdLow> [0,1]
<Units> [0,1]

Tag: <UpdateProperty>
Description: Updates an existing Property for a Class Definition.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Description> [0,1]
<Default> [0,1]
<Writeable> [0,1]
<AlarmThresholdHigh> [0,1]

<AlarmThresholdLow> [0,1]
<Units> [0,1]

Attribute: **Name**
Description: Property name to be updated.
DataType: string
Use: required

Tag: **<DeleteProperty>**
Description: Deletes a Property. Note that this will result in the deletion of the Property in all currently defined Instance Nodes.

Content Type: none
Data Type: N/A

Attribute: **Name**
Description: Property name to be deleted.
DataType: string
Use: required

Tag: **<Name>** (child of <Property>, <AddProperty>, <AlarmAttribute>, <AddAlarmAttribute>)
Description: Name of the Property/AlarmAttribute. This name must be unique within the Class Definition and all of its base classes.
Content Type: textOnly
Data Type: string (max of 64 characters)
Child Elements: none

Tag: **<DataType>**
Description: Data type of the Property. The value of the <DataType> tag should be one of the supported VARENUM values as described in section 1.8.4.
Content Type: textOnly
Data Type: unsignedShort
Child Elements: none

Tag: **<Default>**
Description: Default/initial value of Property. If no value for the Property is sent when an instance of the class is created, this value will be used.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: **<Writeable>**
Description: Boolean that serves as a hint to client applications whether or not the Property can be set from a client application.
Content Type: textOnly
Data Type: Boolean
Default Value: false
Child Elements: none

Tag: <AlarmThresholdHigh>
Description: Alarm threshold value that can be used to implicitly create an Alarm object if the value of the Property exceeds the threshold.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: <AlarmThresholdLow>
Description: Alarm threshold value that can be used to implicitly create an Alarm object if the value of the Property drops below the threshold.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: <Units>
Description: Engineering units for the Property. This is used for client display applications only.
Content Type: textOnly
Data Type: string (max of 32 characters)
Child Elements: none

Tag: <AlarmAttribute> (for <ClassDefintion>)
<AddAlarmAttribute> (for <UpdateClassDefinition>)
Description: Defines a new Alarm Attribute for a Class Definition.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Name> [1,1]
<Description> [0,1]
<Severity> [0,1]
<SingleState> [0,1]
<MsgIDs> [0,1]

Tag: <UpdateAlarmAttribute>
Description: Updates an existing Alarm Attribute for a Class Definition.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Description> [0,1]
<Severity> [0,1]
<SingleState> [0,1]
<MsgIDs> [0,1]

Attribute: Name
Description: AlarmAttribute name to be updated.
DataType: string
Use: required

Tag:	**<Severity>**
Description:	Defines the severity of the Alarm. Values can range from [1,6] with a value of 1 being the most severe. A value of 9 is also allowed and this represents a special type of Alarm referred to as an Application Error. See section 1.8.5 for more details on Severity and Application Errors.
Content Type:	textOnly
Data Type:	unsignedByte
Child Elements:	none

Tag:	**<SingleState>**
Description:	Boolean that indicates if the Alarm is a single-state Alarm object. A single-state Alarm means that only a Raise Event will be sent. No corresponding Clear Event exists. This means it is up to the client Applications to clear the Alarm.
Content Type:	textOnly
Data Type:	boolean
Child Elements:	none

Tag:	**<MsgIDs>**
Description:	Contains a set of MsgIDs that are to be associated with the AlarmAttribute. This element is optional and serves as a cross-reference to the full set of MsgIDs described in a separate XML document.
Content Type:	elementOnly
Data Type:	N/A
Child Elements:	<MsgID> [1,…] <AddMsgID> (for <UpdateAlarmAttribute>) [0,…] <RemoveMsgID> (for <UpdateAlarmAttribute>) [0,…]

Tag:	**<MsgID>** **<AddMsgID>**
Description:	Defines a particular unsigned 32-bit integer MsgID that is associated with the AlarmAttribute or SimpleEvent.
Content Type:	textOnly
Data Type:	unsignedInt
Child Elements:	none

Tag:	**<RemoveMsgID>**
Description:	Defines a particular unsigned 32-bit integer MsgID to be removed from the list of MsgIDs associated with the AlarmAttribute or SimpleEvent.
Content Type:	textOnly
Data Type:	unsignedInt
Child Elements:	none

1.14.8 Examples

This XML sample registers the ICM product:

1.15 MsgID Text

1.15.1 Overview

An ET Provider supplies a set of MsgIDs and their related descriptions as part of defining the set of Class Definitions for a product. In addition, an Enterprise Monitoring Site can supply its own set of MsgIDs and descriptions (or a subset) that can override the descriptions supplied by the ET Provider.

The set of MsgIDs must be sent after the product has been registered since the ProductNamespaceGUID must already have been registered. If this is not done, an error results.

Unlike the <ListenerProductAdmin> XML document, the document that includes MsgIDs can be sent in total or in bits and pieces. One possible scenario is to send the set of MsgIDs for a new language at a different time.

When a set of MsgIDs are sent, if the source of the MsgIDs is from an ET Provider, a first check is made to see if the ProductVersion being sent in the document matches the ProductVersion that has been registered. If this is not the case, the document will be rejected.

There are actually two sets of tables maintained by the ET server software for the set of MsgIDs for any product. One set contains whatever MsgIDs are sent by the ET Provider (the *Provider tables*). This is flagged by the Source='ETProvider' attribute in the <MsgIDs> element. The other set is the "*active*" set which could be either the same as the ETProvider set (the default) or a full or partial override of the MsgIDs via a set of MsgIDs sent from the local Enterprise Monitoring Site or Distributor site. This is flagged by the Source='Local' attribute. The active set of tables can be reset to the "factory" default of using the ET Provider MsgIDs by sending a <MsgIDs Source='Reset'> message.

Whenever a MsgID from an ET Provider (Source='ETProvider') is processed at the Enterprise Monitoring Site, it is merged into these two sets of tables using the following rules:

- If the ProductNamespaceGUID/MsgID/LanguageID does not exist in the Provider tables, it is added to both the Provider tables and the active tables.
- If the ProductNamespaceGUID/MsgID/LanguageID exists in the Provider tables, its current entry is overridden by the new one. If the entry in the active tables is not a local entry, it is also overridden. If the entry is a local entry, then nothing is done to the active tables.

Whenever a MsgID from a local source (Source='Local') is processed at the Enterprise Monitoring Site, the entry always overrides what is in the active tables, but the Provider tables are never touched.

MsgIDs can also be deleted, if necessary, but this action should be a rare occurrence.

To summarize the content of this section, a set of MsgIDs and their associated descriptions can be provided for multiple languages in one XML document. This document has a root tag of <ListenerProductI18N> and can contain the following things (that relate to MsgIDs):

- A set of MsgIDs and their associated descriptions from the ET Provider.
- A set of MsgIDs and their associated descriptions from the Enterprise Monitoring Site. This information overrides what is stored for the ET Provider.
- A reset of any local-provided (from the Enterprise Monitoring Site) MsgID information back to the ones specified by the ET Provider.
- Requests to delete MsgIDs.

1.15.2 Providing MsgID Descriptions

To send a set of MsgIDs to the Listener, an XML document with a root tag of <ListenerProductI18N> is sent via the HTTP POST method to the following URN:

`https://ListenerURL/ETListener/ETListener.dll?ListenerMessages`

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used for this document type:

```
<ListenerProductI18N ETVersion='3.0'>

  <MsgIDs ProductNamespaceGUID='guid' ProductVersion='product version'
Source='ETProvider, Local or Reset'>
    <MsgID Value='32-bit unsigned integer value'>
      <Language ID='id'>
        <MsgText>message text with substitution arguments</MsgText>
        <Description>description of what the Event
means</Description>
        <Action>information on what a support organization should do
when this Event occurs</Action>
      </Language>
    </MsgID>

    <DeleteMsgID Value='32-bit unsigned integer value' />

  </MsgIDs>

</ListenerProductI18N>
```

As previously stated, the ProductNamespaceGUID must have already been registered at the Enterprise Monitoring Site and the ProductVersion number must match the one

currently registered. If either of these conditions are not met, an error is returned and nothing is processed.

The Source attribute identifies the source of the MsgID data and defines where the messages are to be placed. See section 1.15.1 for the rules on how the messages are processed.

This type of document can contain exactly one <MsgIDs> element and an arbitrary number of <MsgID> and <DeleteMsgID> elements. Similarly, each <MsgID> element can contain an arbitrary number of <Language> elements.

The Listener's response to the above document is in the following form:

```
<ListenerProductI18NResponse ETVersion='3.0'>
  <MsgIDs ProductNamespaceGUID='guid' Status='HRESULT'>
  Error text if HRESULT is non zero
    <MsgID Value='32-bit unsigned integer value' Status='HRESULT'>
     Error text if HRESULT is non zero
    </MsgID>
    <DeleteMsgID Value='32-bit unsigned integer value'
Status='HRESULT'>
     Error text if HRESULT is non zero
    </DeleteMsgID>
  </MsgIDs>
</ListenerProductI18NResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the document is successfully processed, the <MsgIDs> Status value will be zero (0), and the element will have no child elements. If the Status value is non-zero, some kind of error occurred.

If the error occurred in the processing of a specific <MsgID> element, then the document will contain one or more <MsgID> elements that describe each error that occurred. The <ListenerProductI18N> document may result in a partial success. That is, each <MsgID> element is processed as an individual transaction, not the document as a whole. Thus, if there are errors (other than schema errors which would reject the whole document), it is possible that some of the <MsgID>s are successfully processed, and some are not. The ones that result in errors will be returned as described above.

The following error values have been defined for this type of document:

Status Values		
Value	Mnemonic	Error Text

0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[ProductNamespaceGUID],[MsgID]' does not exist.
0x8022000A (-2145255416)	MSGMGR_E_PRODUCTVERSIONMISMATCH	The product Version number does not match what is currently registered (%1).

If the <MsgIDs> Status value is MSGMGR_E_KEYDOESNOTEXIST, this means that the product has not been registered yet. The rest of the document will not contain any <MsgID> elements.

If the <MsgIDs> ProductVersion in the source document does not match what is currently registered, the <MsgIDs> Status value is MSGMGR_E_PRODUCTVERSIONMISMATCH. The rest of the document will not contain any <MsgID> elements.

If the <MsgIDs> Status value is ET_E_ERRORSCONTAINED, it means that at least one of the <MsgID> elements resulted in some kind of error. In this case, the document will contain the list of <MsgID>s that resulted in an error, with a Status value on each one indicating the error that occurred.

If an attempt is made to delete a MsgID that does not exist, the <DeleteMsgID> Status value is MSMGR_E_KEYDOESNOTEXIST.

1.15.3 XML Schemas

The following XML Schema Document **ListenerProductI18N.xsd** describes the schema for the <ListenerProductI18N> document:

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:tns='urn:www.cisco.com:ListenerProductI18N'
targetNamespace='urn:www.cisco.com:ListenerProductI18N'>

<xsd:element name='ListenerProductI18N'>
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name='MsgIDs'>
        <xsd:complexType>
          <xsd:sequence minOccurs='1' maxOccurs='unbounded'>
            <xsd:element name='MsgID' minOccurs='0'
maxOccurs='unbounded'>
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name='Language' minOccurs='1'
maxOccurs='unbounded'>
```

```
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name='MsgText'
type='xsd:string' />
                            <xsd:element
name='Description' type='xsd:string' minOccurs='0' />
                            <xsd:element name='Action'
type='xsd:string' minOccurs='0' />
                        </xsd:sequence>
                        <xsd:attribute name='ID'
type='xsd:unsignedShort' use='required'/>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name='Value' type='xsd:unsignedInt'
use='required'/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name='DeleteMsgID' minOccurs='0'
maxOccurs='unbounded'>
        <xsd:complexType>
            <xsd:attribute name='Value' type='xsd:unsignedInt'
use='required'/>
        </xsd:complexType>
    </xsd:element>
</xsd:sequence>
<xsd:attribute name='ProductNamespaceGUID' type='tns:GUID'
use='required' />
<xsd:attribute name='ProductVersion' type='xsd:float'
use='required'/>
<xsd:attribute name='Source' type='MessageSource'
use='required'/>
</xsd:complexType>
</xsd:element>
<xsd:element name='Descriptions'>
    <xsd:complexType>
        <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
            <xsd:element name='ProductDescription' minOccurs='0'
maxOccurs='1'>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name='Language' minOccurs='1'
maxOccurs='unbounded'>
                            <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element
name='Description' type='xsd:string' />
                                </xsd:sequence>
                                <xsd:attribute name='ID'
type='xsd:unsignedShort' use='required'/>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
```

```
                </xsd:element>
                <xsd:element name='ClassDescription' minOccurs='0'
maxOccurs='unbounded'>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name='Language' minOccurs='1'
maxOccurs='unbounded'>
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element
name='Description' type='xsd:string' />
                                    </xsd:sequence>
                                    <xsd:attribute name='ID'
type='xsd:unsignedShort' use='required'/>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name='PropertyDescription' minOccurs='0'
maxOccurs='unbounded'>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name='Language' minOccurs='1'
maxOccurs='unbounded'>
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element
name='Description' type='xsd:string' />
                                    </xsd:sequence>
                                    <xsd:attribute name='ID'
type='xsd:unsignedShort' use='required'/>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                        <xsd:attribute name='Name' type='xsd:string'
use='required'/>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name='AlarmAttributeDescription'
minOccurs='0' maxOccurs='unbounded'>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name='Language' minOccurs='1'
maxOccurs='unbounded'>
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element
name='Description' type='xsd:string' />
                                    </xsd:sequence>
```

```
                                        <xsd:attribute name='ID'
type='xsd:unsignedShort' use='required'/>
                                    </xsd:complexType>
                                </xsd:element>
                            </xsd:sequence>
                            <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                            <xsd:attribute name='Name' type='xsd:string'
use='required'/>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name='SimpleEventDescription' minOccurs='0'
maxOccurs='unbounded'>
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name='Language' minOccurs='1'
maxOccurs='unbounded'>
                                    <xsd:complexType>
                                        <xsd:sequence>
                                            <xsd:element
name='Description' type='xsd:string' />
                                        </xsd:sequence>
                                        <xsd:attribute name='ID'
type='xsd:unsignedShort' use='required'/>
                                    </xsd:complexType>
                                </xsd:element>
                            </xsd:sequence>
                            <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                            <xsd:attribute name='Name' type='xsd:string'
use='required'/>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name='DeleteProductDescription'
minOccurs='0' />
                    <xsd:element name='DeleteClassDescription' minOccurs='0'
maxOccurs='unbounded'>
                        <xsd:complexType>
                            <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name='DeletePropertyDescription'
minOccurs='0' maxOccurs='unbounded'>
                        <xsd:complexType>
                            <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                            <xsd:attribute name='Name' type='xsd:string'
use='required'/>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name='DeleteAlarmAttributeDescription'
minOccurs='0' maxOccurs='unbounded'>
                        <xsd:complexType>
```

```
                    <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                    <xsd:attribute name='Name' type='xsd:string'
use='required'/>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name='DeleteSimpleEventDescription'
minOccurs='0' maxOccurs='unbounded'>
                <xsd:complexType>
                    <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required'/>
                    <xsd:attribute name='Name' type='xsd:string'
use='required'/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name='ProductNamespaceGUID' type='tns:GUID'
use='required' />
        <xsd:attribute name='ProductVersion' type='xsd:float'
use='required'/>
        <xsd:attribute name='Source' type='MessageSource'
use='required'/>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
  <xsd:attribute name='ETVersion' type='xsd:float' use='required'/>
 </xsd:complexType>
</xsd:element>

<xsd:simpleType name='MessageSource'>
  <xsd:restriction base='xsd:string'>
    <xsd:enumeration value='ETProvider' />
    <xsd:enumeration value='Local' />
    <xsd:enumeration value='Reset' />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name='GUID'>
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value='[0-9A-Fa-f]{8}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{4}-
[0-9A-Fa-f]{4}-[0-9A-Fa-f]{12}' />
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

1.15.4 XML Tag Descriptions

The following XML tags are defined in the <ListenerProductI18N> document that are relevant to MsgIDs:

Tag: <ListenerProductI18N> (root element)

Description: Defines the root element for the ListenerProductI18N document. This type of document contains MsgID text and description elements, or Descriptions of Class Definitions, Properties, Alarm Attributes and Simple Events.
Content Type: elementOnly
Data Type: N/A
Child Elements: <MsgIDs> [0,1]
<Descriptions> [0,1] (see section 1.16)

Attribute: ETVersion
Description: ET protocol version number of document. Must be 3.0.
DataType: float
Use: required
Value: 3.0

Tag: <MsgIDs>
Description: Contains a set of MsgIDs for a given product.
Content Type: elementOnly
Data Type: N/A
Child Elements: <MsgID> [0,…]
<DeleteMsgID> [0,…]

Attribute: ProductNamespaceGUID
Description: GUID that uniquely identifies this product.
DataType: uuid
Use: required

Attribute: ProductVersion
Description: Product Version for these MsgIDs.
DataType: float
Use: required

Attribute: Source
Description: Source of MsgIDs. The source could be from an ETProvider (base definitions), a Local source that provide an override to the base definitions, or it could be a Reset flag that reverts the MsgID descriptions to the ET Provider defaults.
DataType: string (one of 'ETProvider', 'Local' or 'Reset')
Use: required

Tag: <MsgID>
Description: Information about a particular MsgID.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Language> [1,…]

Attribute: Value
Description: Defines a particular unsigned 32-bit integer MsgID that is associated with the AlarmAttribute or SimpleEvent.
DataType: unsignedInt
Use: required

Tag: **<Language>**
Description: Contains MsgID information for a particular language.
Content Type: elementOnly
Data Type: N/A
Child Elements: <MsgText> [1,1]
<Description> [0,1]
<Action> [0,1]

Attribute: ID
Description: Language ID for the child Elements of this element. See Appendix B for a table of Language Identifiers and more information about Language Identifiers.
DataType: unsignedShort
Use: required

Tag: **<MsgText>**
Description: Message text with substitution parameters (%1, %2, etc.) that describe the MsgID. For more information on how substitution parameters are used, see section 1.9.1.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: **<Description>**
Description: This is a more verbose and general description of the meaning and significance of this Event in the language specified. This description might be shown by client application users that want more detail on the meaning of this particular Event.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: **<Action>**
Description: This is text in the language specified that describes what action (if any) a support center representative might take to remedy the problem.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: **<DeleteMsgID>**
Description: Deletes a particular MsgID from the tables.
Content Type: empty
Data Type: N/A
Child Elements: none

Attribute: Value
Description: Specifies the particular unsigned 32-bit integer MsgID that is associated with the AlarmAttribute or SimpleEvent.
DataType: unsignedInt

Use: required

1.15.5 Examples
TBD

1.16 Descriptions

1.16.1 Overview
When you define various objects via the ET protocols, you provide <Description> elements which are text fields that describe what the particular object is. By design, these Descriptions are provided in English.

The ET specification allows an ET Provider to supply these Descriptions in multiple languages to support I18N efforts. It also allows an Enterprise Monitoring Site to supply its own set of Descriptions (either in English or in any set of languages) to override the Descriptions defined by the ET Provider.

The ET specification allows you to provide text Description fields associated with the following objects:

- Product
- Class Definition
- Property of a Class Definition
- Alarm Attribute
- Simple Event

By convention when you define these objects via the ET protocols described elsewhere in section 1.12.5, you provide <Description> text elements in English (Language Identifier = 1033). The ET specifications allow you to provide these Descriptions in other languages so that client applications that may want to describe this information can see the text in the language of their choice.

The set of Descriptions must be sent after the product has been registered since the ProductNamespaceGUID must already have been registered. If this is not done, an error results.

Unlike the <ListenerProductAdmin> XML document, the document that includes Descriptions can be sent in total or in bits and pieces. One possible scenario is to send the set of Descriptions for a new language at a different time.

When a set of Descriptions are sent, if the source of the Descriptions is from an ET Provider, a first check is made to see if the ProductVersion being sent in the document

matches the ProductVersion that has been registered. If this is not the case, the document will be rejected.

There are actually two sets of tables maintained by the ET server software for the set of Descriptions for any product. One set contains whatever Descriptions are sent by the ET Provider (the *Provider tables*). This is flagged by the Source='ETProvider' attribute in the <Descriptions> element. The other set is the "*active*" set which could be either the same as the ETProvider set (the default) or a full or partial override of the Descriptions via a set of Descriptions sent from the local Enterprise Monitoring Site or Distributor site. This is flagged by the Source='Local' attribute. The active set of tables can be reset to the "factory" default of using the ET Provider Descriptions by sending a <Descriptions Source='Reset'> message.

Whenever a Description from an ET Provider (Source='ETProvider') is processed at the Enterprise Monitoring Site, it is merged into these two sets of tables using the following rules:

- If the ProductNamespaceGUID/MsgID/LanguageID does not exist in the Provider tables, it is added to both the Provider tables and the active tables.
- If the ProductNamespaceGUID/Description/LanguageID exists in the Provider tables, its current entry is overridden by the new one. If the entry in the active tables is not a local entry, it is also overridden. If the entry is a local entry, then nothing is done to the active tables.

Whenever a Description from a local source (Source='Local') is processed at the Enterprise Monitoring Site, the entry always overrides what is in the active tables, but the Provider tables are never touched.

Descriptions can also be deleted, if necessary, but this action should be a rare occurrence.

1.16.2 Providing Descriptions

To send a set of Descriptions to the Listener, an XML document with a root tag of <ListenerProductI18N> is sent via the HTTP POST method to the following URN:

`https://ListenerURL/ETListener/ETListener.dll?ListenerMessages`

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used for this document type:

```
<ListenerProductI18N ETVersion='3.0'>
  <Descriptions ProductNamespaceGUID='guid' ProductVersion='product
version' Source='ETProvider, Local or Reset'>
    <ProductDescription>
      <Language ID='id'>
        <Description>description of the product</Description>
```

```
    </Language>
  </ProductDescription>
  <ClassDescription OID='oid'>
    <Language ID='id'>
      <Description>description of the class</Description>
    </Language>
  </ClassDescription>
  <PropertyDescription OID='oid' Name='property name'>
    <Language ID='id'>
      <Description>description of the Property</Description>
    </Language>
  </PropertyDescription>
  <AlarmAttributeDescription OID='oid' Name='Alarm Attribute name'>
    <Language ID='id'>
      <Description>description of the Alarm Attribute</Description>
    </Language>
  </AlarmAttributeDescription>

  <DeleteProductDescription />

  <DeleteClassDescription OID='oid' />

  <DeletePropertyDescription OID='oid' Name='property name' />

  <DeleteAlarmAttributeDescription OID='oid' Name='Alarm Attribute
name' />

 </Descriptions>

</ListenerProductI18N>
```

As previously stated, the ProductNamespaceGUID must have already been registered at the Enterprise Monitoring Site and the ProductVersion number must match the one currently registered. If either of these conditions are not met, an error is returned and nothing is processed.

The Source attribute identifies the source of the data and defines where the Descriptions are to be placed. See section 1.16.1 for the rules on how the messages are processed.

This type of document can contain exactly one <MsgIDs> element and an arbitrary number of <MsgID> and <DeleteMsgID> elements. Similarly, each <MsgID> element can contain an arbitrary number of <Language> elements.

The Listener's response to the above document is in the following form:

```
<ListenerProductI18NResponse ETVersion='3.0'>

  <Descriptions ProductNamespaceGUID='guid' Status='HRESULT'>
   Error text if HRESULT is non zero
    <ProductDescription Status='HRESULT'>
     Error text if HRESULT is non zero
    </ProductDescription>
```

```
    <ClassDescription OID='oid' Status='HRESULT'>
      Error text if HRESULT is non zero
    </ClassDescription>

    <PropertyDescription OID='oid' Name='property name'
Status='HRESULT'>
      Error text if HRESULT is non zero
    </PropertyDescription>

    <AlarmAttributeDescription OID='oid' Name='Alarm Attribute name'
Status='HRESULT'>
      Error text if HRESULT is non zero
    </AlarmAttributeDescription>

    <DeleteProductDescription  Status='HRESULT'>
      Error text if HRESULT is non zero
    </DeleteProductDescription>

    <DeleteClassDescription OID='oid' Status='HRESULT'>
      Error text if HRESULT is non zero
    </DeleteClassDescription>

    <DeletePropertyDescription OID='oid' Name='property name'
Status='HRESULT'>
      Error text if HRESULT is non zero
    </DeletePropertyDescription>

    <DeleteAlarmAttributeDescription OID='oid' Name='Alarm Attribute
name' Status='HRESULT'>
      Error text if HRESULT is non zero
    </DeleteAlarmAttributeDescription>

  </Descriptions>
</ListenerProductI18NResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the document is successfully processed, the <Descriptions> Status value will be zero (0), and the element will have no child elements. If the Status value is non-zero, some kind of error occurred.

If the error occurred in the processing of a specific Description element, then the document will contain one or more <*XXX*Description> elements that describe each error that occurred. The <ListenerProductI18N> document may result in a partial success. That is, each <*XXX*Description> element is processed as an individual transaction, not the document as a whole. Thus, if there are errors (other than schema errors which would reject the whole document), it is possible that some of the <*XXX*Description>s

are successfully processed, and some are not. The ones that result in errors will be returned as described above.

The following error values have been defined for this type of document:

Status Values		
Value	Mnemonic	Error Text
	ET E V S EM	n ali sc ema
x	ET E E S T E	Some errors exist in t e c il elements
x	MS M E E ES TE ST	ro uct ames ace ame oes not exist
x	MS M E TVE S M SM T	T e ro uct Version num er oes not matc at is currentl re istere

If the <Descriptions> Status value is MSGMGR_E_KEYDOESNOTEXIST, this means that the product has not been registered yet. The rest of the document will not contain any <*XXX*Description> elements.

If the <Descriptions> ProductVersion in the source document does not match what is currently registered, the <Descriptions> Status value is MSGMGR_E_PRODUCTVERSIONMISMATCH. The rest of the document will not contain any <*XXX*Description> elements.

If the <Descriptions> Status value is ET_E_ERRORSCONTAINED, it means that at least one of the <*XXX*Description> elements resulted in some kind of error. In this case, the document will contain the list of <*XXX*Description>s that resulted in an error, with a Status value on each one indicating the error that occurred.

If an attempt is made to delete a Description that does not exist, the Status value is MSMGR_E_KEYDOESNOTEXIST.

1.16.3 XML Schemas

The Descriptions messages share the same XML document type <ListenerProductI18N> root tag element and, therefore, share the same XML Schema Definition. This schema is shown in section 1.15.3.

1.16.4 XML Tag Descriptions

The following XML tags are defined in thhe <ListenerProductI18N> documen that are relevant to Descriptions:

Tag: <ListenerProductI18N> (root element)

Description: Defines the root element for the ListenerProductI18N document. This type of document contains MsgID text and description elements, or Descriptions of Class Definitions, Properties, Alarm Attributes and Simple Events.
Content Type: elementOnly
Data Type: N/A
Child Elements: <MsgIDs> [0,1] (see section 1.15.4)
<Descriptions> [0,1]

Attribute: **ETVersion**
Description: ET protocol version number of document. Must be 3.0.
DataType: float
Use: required
Value: 3.0

Tag: **<Descriptions>**
Description: Contains a set of Descriptions for a given product.

Content Type: elementOnly
Data Type: N/A
Child Elements: <ProductDescription> [0,1]
<ClassDescription> [0,…]
<PropertyDescription> [0,…]
<AlarmAttributeDescription> [0,…]
<DeleteProductDescription> [0,1]
<DeleteClassDescription> [0,…]
<DeletePropertyDescription> [0,…]
<DeleteAlarmAttributeDescription> [0,…]

Attribute: **ProductNamespaceGUID**
Description: GUID that uniquely identifies this product.
DataType: uuid
Use: required

Attribute: **ProductVersion**
Description: Product Version for these Descriptions. (Ignored when Source='Local' or 'Reset'.)
DataType: float
Use: required

Attribute: **Source**
Description: Source of MsgIDs. The source could be from an **ETProvider** (base definitions), a **Local** source that provide an override to the base definitions, or it could be a **Reset** flag that reverts the MsgID descriptions to the ET Provider defaults.
DataType: string (one of 'ETProvider', 'Local' or 'Reset')
Use: required

Tag: **<ProductDescription>**
Description: Description for the product specified for the ProductNamespaceGUID.
Content Type: elementOnly
Data Type: N/A

Child Elements: <Language> [1,…]

Tag: <ClassDescription>
Description: Description for a particular Class Definition.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Language> [1,…]

Attribute: OID
Description: Defines a particular Class Definition within the product namespace that is to be associated with this Description.
DataType: unsignedInt
Use: required

Tag: <PropertyDescription>
Description: Description for a particular Property of a class.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Language> [1,…]

Attribute: OID
Description: Defines a particular Class Definition within the product namespace that is to be associated with this Description.
DataType: unsignedInt
Use: required

Attribute: Name
Description: Property of the class associated with this Description.
DataType: string
Use: required

Tag: <AlarmAttributeDescription>
Description: Description for a particular Alarm Attribute of a class.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Language> [1,…]

Attribute: OID
Description: Defines a particular Class Definition within the product namespace that is to be associated with this Description.
DataType: unsignedInt
Use: required

Attribute: Name
Description: Alarm Attribute of the class assocciated with this Description.
DataType: string
Use: required

Tag:	**<Language>**
Description:	Contains Description information for a particular language.
Content Type:	elementOnly
Data Type:	N/A
Child Elements:	<Description> [1,1]
Attribute:	**ID**
Description:	Language ID for the Description. See Appendix B for a table of Language Identifiers and more information about Language Identifiers.
DataType:	unsignedShort
Use:	required

Tag:	**<Description>**
Description:	The description of the particular object.
Content Type:	textOnly
Data Type:	string (max of 256 characters)
Child Elements:	none

Tag:	**<DeleteProductDescription>**
Description:	Deletes the Description for the product specified.
Content Type:	empty
Data Type:	N/A
Child Elements:	none

Tag:	**<DeleteClassDescription>**
Description:	Deletes the Description for a particular Class Definition.
Content Type:	empty
Data Type:	N/A
Child Elements:	none
Attribute:	**OID**
Description:	Defines a particular Class Definition within the product namespace.
DataType:	unsignedInt
Use:	required

Tag:	**<DeletePropertyDescription>**
Description:	Deletes the Description for a particular Property of a class.
Content Type:	empty
Data Type:	N/A
Child Elements:	none
Attribute:	**OID**
Description:	Defines a particular Class Definition within the product namespace.
DataType:	unsignedInt
Use:	required

```
Attribute:      Name
Description:    Property of the class.
DataType:       string
Use:            required

Tag:            <DeleteAlarmAttributeDescription>
Description:    Deletes the Description for a particular Alarm
Attribute of a class.
Content Type:   empty
Data Type:      N/A
Child Elements: none

Attribute:      OID
Description:    Defines a particular Class Definition within the
product namespace.
DataType:       unsignedInt
Use:            required

Attribute:      Name
Description:    Alarm Attribute of the class.
DataType:       string
Use:            required
```

1.16.5 Examples

TBD

1.17 Site Registration

1.17.1 Overview

Registering a site is the first type of instance object that you can create. Once a site has been registered, you can begin sending data which include sending Instance Nodes declarations, Property updates and Alarm Events.

As described in section 1.9, site-related messages consist of the following four types:

- <RegisterSite> - registers a Site description to the Enterprise Monitoring Site's Site list. The <RegisterSite> message must be the first site-related message sent. This message defines a new site as a source of ET data.
- <SiteOnline> - indicates that a Site is now online. When a Site is online, it means that it will begin sending data and will adhere to heartbeat reporting requirements specified in the <SiteOnline> message.
- <SiteOffline> - indicates that a Site is now offline. This means that it will cease sending Listener Messages and will no longer adhere to heartbeat reporting requirements.

- <DeleteSite> deletes a Site. This message should be used carefully since it results in the deletion of a Site and all of its associated objects at the Enterprise Monitoring Site. This means that all Instance Node and archived Alarm data will be deleted from all databases.

These four message types are described in this section.

1.17.2 Registering a Site

To register a new site, an ET Provider sends a <RegisterSite> message in an XML document to Listener with a root tag of <ListenerSiteAdmin>. The XML document must be sent via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListener/ETListener.dll?ListenerMessages
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to register a new site at an Enterprise Monitoring Site:

```
<ListenerSiteAdmin ETVersion='3.0'>
  <RegisterSite SiteType='Enterprise or Customer' SiteGUID='guid'
SiteName='name'>
    <DownStreamSiteGUID>guid [optional]</DownStreamSiteGUID>
  </RegisterSite>
</ListenerSiteAdmin>
```

A <ListenerSiteAdmin> document must contain exactly one child element, of which <RegisterSite> is one of the allowable types.

The SiteType attribute indicates the type of site that is being registered. If the SiteType='Enterprise', then additional child elements are required. Note that for an ET Provider, you should not have to worry about registering upstream Enterprise Monitoring Sites. This is a responsibility of the ET server software. This description is included here for completeness.

In addition, if Enterprise Monitoring Sites are chained as illustrated in Figure 2, the <RegisterSite> message may need to include the <DownStreamSiteGUID> tag if the site being registered is further upstream than the one sending the message. Again, this is the responsibility of the ET server software to send this information. It is not necessary for an ET Provider to worry about this.

The Listener's response to a <RegisterSite> message is in the following form:

```
<ListenerSiteAdminResponse ETVersion='3.0'>
  <RegisterSite SiteGUID='guid' Status='HRESULT'>Error text if HRESULT
is non zero</RegisterSite>
```

```
</ListenerSiteAdminResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80220001 (-2145255423)	MSGMGR_E_KEYALREADYEXISTS	%1 is not unique. It has already been assigned to %2.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	%1 does not exist.

The MSGMGR_E_KEYDOESNOTEXIST error may be returned if the DownstreamSiteGUID has not been previously registered.

1.17.3 Putting a Site Online

To put a site online, an ET Provider sends a <SiteOnline> message in an XML document to Listener with a root tag of <ListenerSiteAdmin>. The XML document must be sent via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListener/ETListener.dll?ListenerMessages
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to put a site online at an Enterprise Monitoring Site:

```
<ListenerSiteAdmin Version='3.0'>

  <SiteOnline SiteGUID='guid'>
     <ListenerAURL>URL [valid only for
SiteType='Enterprise']</ListenerAURL>
     <ListenerBURL>URL [valid only for
SiteType='Enterprise']</ListenerBURL>
     <HeartbeatInterval Source='source name' Interval='interval in
minutes' />
     <ReadOnly>True or False [optional, default is False]</ReadOnly>
  </SiteOnline>
```

```
</ListenerSiteAdmin>
```

A <ListenerSiteAdmin> document must contain exactly one child element, of which <SiteOnline> is one of the allowable types.

For an Enterprise site, it must also send the Listener URLs to the downstream Enterprise Monitoring Site if the ReadO so that it will be able to forward user actions that pertain to it from the downstream site. These Listener URLs are sent via the <ListenerAURL> and <ListenerBURL> tags. Note that for an ET Provider, you should not have to worry about registering upstream Enterprise Monitoring Sites. This is a responsibility of the ET server software. This description is included here for completeness.

The <HeartbeatInterval> element defines the interval in which Heartbeat messages will be sent to the Listener URLs. You must register at least one Source, and you can register up to two Sources if you are doing fault-tolerant processing. The Source names distinguish the two sources. The Interval value is measured in minutes. For more information on Heartbeat message processing see sections 1.12.5, 0 and 1.22.

The <ReadOnly> element is a directive that tells the Enterprise Monitoring Site that it can or cannot perform User Actions on Alarm objects. If <ReadOnly> is False (the default value used if not present), then client applications can perform User Actions on Alarm objects. If <ReadOnly> is True, then User Actions will not be allowed on these objects.

Note that when a Site goes online, it implicitly puts all upstream Enterprise Monitoring Sites online as well. This is because they are passing data through to this Site so their data will implicitly flow through as well. Thus, there is no need for an Enterprise Site to send additional <SiteOnline> messages for any upstream Enterprise Monitoring Sites. Again, an ET Provider should not have to worry about this. This is taken care of by the ET server software.

The Listener's response to a <SiteOnline> message is in the following form:

```
<ListenerSiteAdminResponse Version='3.0'>
    <SiteOnline SiteGUID='guid' Status='HRESULT'>
    Error text if HRESULT is non zero
    </SiteOnline>
</ListenerSiteAdminResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**

0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	%1 does not exist.
0x80220007 (-2145255419)	MSGMGR_E_SITEALREADYONLINE	The Site is already online.

1.17.4 Putting a Site Offline

To put a site onffline, an ET Provider sends a <SiteOffline> message in an XML document to Listener with a root tag of <ListenerSiteAdmin>. The XML document must be sent via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to put a site offline at an Enterprise Monitoring Site:

```
<ListenerSiteAdmin Version='3.0'>
  <SiteOffline SiteGUID='guid'>
    <Notes>text information to indicate why the site is going offline
[optional]</Notes>
  </SiteOffline>
</ListenerSiteAdmin>
```

A <ListenerSiteAdmin> document must contain exactly one child element, of which <SiteOffline> is one of the allowable types.

The <Notes> element is optional and can be used for client applications at the Enterprise Monitoring Site to display the reason why the Site is going offline.

Note that when a Site goes offline, it implicitly puts all upstream Enterprise Monitoring Sites offline as well. This is because they are passing data through to this Site so their data will implicitly stop flowing through as well. Thus, there is no need for an Enterprise Site to send additional <SiteOffline> messages for any upstream Enterprise Monitoring Sites. Again, an ET Provider should not have to worry about this. This is taken care of by the ET server software.

The Listener's response to a <SiteOffline> message is in the following form:

```
<ListenerSiteAdminResponse Version='3.0'>
  <SiteOffline SiteGUID='guid' Status='HRESULT'>
```

```
  Error text if HRESULT is non zero
  </SiteOffline>
</ListenerSiteAdminResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	%1 does not exist.
0x80220008 (-2145255418)	MSGMGR_E_SITEALREADYOFFLINE	The Site is already offline.

1.17.5 Site Deletion

To delete a Site, the sender must be a member of the ETAdmin user group. To delete a Site, an administrator (presumably at an Enterprise Monitoring Site) sends a <DeleteSite> message in an XML document to Listener with a root tag of <ListenerSiteAdmin>. The XML document must be sent via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListenerInt/ETListener.dll?ListenerAdmin
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to delete a site at an Enterprise Monitoring Site:

```
<ListenerSiteAdmin Version='3.0'>
  <DeleteSite SiteGUID='guid' />
</ListenerSiteAdmin>
```

A <ListenerSiteAdmin> document must contain exactly one child element, of which <DeleteSite> is one of the allowable types.

The Listener's response to a <DeleteSite> message is in the following form:

```
<ListenerSiteAdminResponse Version='3.0'>
  <DeleteSite SiteGUID='guid' Status='HRESULT'>
  Error text if HRESULT is non zero
  </DeleteSite>
</ListenerSiteAdminResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	%1 does not exist.

1.17.6 XML Schemas

The following XML Schema Document **ListenerSiteAdmin.xsd** describes the schema for the <ListenerSiteAdmin> document:

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:tns='http://cisco.com/ListenerSiteAdmin'
targetNamespace='http://cisco.com/ListenerSiteAdmin'>

<xsd:element name='ListenerSiteAdmin'>
    <xsd:complexType>
        <xsd:choice>
            <xsd:element name='RegisterSite'>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name='DownstreamSiteGUID' minOccurs='0'
type='tns:GUID' />
                    </xsd:sequence>
                    <xsd:attribute name='SiteType' type='tns:SiteType'
use='required' />
                    <xsd:attribute name='SiteGUID' type='tns:GUID'
use='required' />
                    <xsd:attribute name='SiteName' type='xsd:string'
use='required' />
                </xsd:complexType>
            </xsd:element>
            <xsd:element name='SiteOnline'>
                <xsd:complexType>
```

```
            <xsd:sequence>
                <xsd:element name='ListenerAURL' minOccurs='0'
type='xsd:string' />
                <xsd:element name='ListenerBURL' minOccurs='0'
type='xsd:string' />
                <xsd:element name='HeartbeatInterval' minOccurs='1'
maxOccurs='2'>
                    <xsd:complexType>
                        <xsd:attribute name='Source' type='xsd:string'
use='required' />
                        <xsd:attribute name='Interval'
type='xsd:unsignedInt' use='required' />
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name='ReadOnly' minOccurs='0'
type='xsd:boolean' />
            </xsd:sequence>
            <xsd:attribute name='SiteGUID' type='tns:GUID'
use='required' />
        </xsd:complexType>
    </xsd:element>
    <xsd:element name='SiteOffline'>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name='Notes' minOccurs='0'
type='xsd:string' />
            </xsd:sequence>
            <xsd:attribute name='SiteGUID' type='tns:GUID'
use='required' />
        </xsd:complexType>
    </xsd:element>
    <xsd:element name='DeleteSite'>
        <xsd:complexType>
            <xsd:attribute name='SiteGUID' type='tns:GUID'
use='required' />
        </xsd:complexType>
    </xsd:element>
  </xsd:choice>
  <xsd:attribute name='ETVersion' type='xsd:float' use='required'/>
 </xsd:complexType>
</xsd:element>

<xsd:simpleType name='SiteType'>
  <xsd:restriction base='xsd:string'>
    <xsd:enumeration value='Enterprise' />
    <xsd:enumeration value='Customer' />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name='GUID'>
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value='[0-9A-Fa-f]{8}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{4}-
[0-9A-Fa-f]{4}-[0-9A-Fa-f]{12}' />
  </xsd:restriction>
```

```
</xsd:simpleType>
</xsd:schema>
```

1.17.7 XML Tag Descriptions

The following XML tags are defined in the <ListenerSiteAdmin> document:

```
Tag:                <ListenerSiteAdmin> (root element)
Description:        Defines the root element for the ListenerSiteAdmin
document. This type of document contains state information related to
sites.
Content Type:       elementOnly
Data Type:          N/A
Child Elements:     <RegisterSite> [0,1]
                    <SiteOnline> [0,1]
                    <SiteOffline> [0,1]
                    <DeleteSite> [0,1]

Attribute:          ETVersion
Description:        ET protocol version number of document. Must be 3.0.
DataType:           float
Use:                required
Value:              3.0
```

```
Tag:                <RegisterSite>
Description:        Registers a new site with the Enterprise Monitoring
Site.
Content Type:       none
Data Type:          N/A
Child Elements:     <DownstreamSiteGUID> [0,1]

Attribute:          SiteType
Description:        Type of site being registered. The choices are
'Enterprise' or 'Customer'.
DataType:           string
Use:                required

Attribute:          SiteGUID
Description:        GUID that uniquely identifies the site.
DataType:           uuid
Use:                required

Attribute:          SiteName
Description:        Display name to be used for the site.
DataType:           string
Use:                required
```

```
Tag:                <DownstreamSiteGUID>
Description:        Site GUID of a previously registered site that is
immediately downstream from the site being registered. (Used for
PassThru sites only.)
Content Type:       textOnly
Data Type:          uuid
Child Elements:     none
```

Tag: <SiteOnline>
Description: Marks a previously registered site as being online. This means the site will begin sending dynamic data and will adhere to the requirements for sending Heartbeat data to the Enterprise Monitoring Site.
Content Type: none
Data Type: N/A
Child Elements: <ListenerAURL> [0,1]
<ListenerBURL> [0,1]
<HeartbeatInterval> [1,2]
<ReadOnly> [0,1]

Attribute: SiteGUID
Description: GUID that uniquely identifies the site.
DataType: uuid
Use: required

Tag: <ListenerAURL>
<ListenerBURL>
Description: URL for the A/B side Listener at this site. The URL should be in the form 'listenera.companyxyz.com'. This element should only be used if the registered site is of SiteType='Enterprise' which marks it as an Enterprise Monitoring Site.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: <HeartbeatInterval>
Description: Heartbeat interval in minutes that indicates how often the site will send Heartbeat messages. See section 1.12.5 for more information on Heartbeat messages.
Content Type: empty
Data Type: N/A
Child Elements: none

Attribute: Source
Description: A display name that indicates who is sending the Heartbeat message.
DataType: string
Use: required

Attribute: Interval
Description: The interval in minutes that indicates how often the source will send Heartbeat messages.
DataType: unsignedInt
Use: required

Tag: <ReadOnly>
Description: Boolean variable that indicates whether or not the site allows Enterprise Monitoring Sites to perform User Actions on Alarm objects. If this element is not present, the default value is **false**.

```
Content Type:     textOnly
Data Type:        boolean
Child Elements:   none

Tag:              <SiteOffline>
Description:      Marks a site as being offline.  This means the site
will cease reporting dynamic data and will no longer send Heartbeat
data to the Enterprise Monitoring Site.
Content Type:     none
Data Type:        N/A
Child Elements:   <Notes> [0,1]

Attribute:        SiteGUID
Description:      GUID that uniquely identifies the site.
DataType:         uuid
Use:              required

Tag:              <Notes>
Description:      Optional notes field that can be used to report the
reason why the site is going offline.  These notes can be displayed to
client applications.
Content Type:     textOnly
Data Type:        string
Child Elements:   none

Tag:              <DeleteSite>
Description:      Permanently deletes a site.  This will result in the
deletion of all data associated with this site.  This includes all
Instance Nodes and all Alarm history data for these Instance Nodes.
Note that only users in the ETAdmin group can issue this type of
message.
Content Type:     empty
Data Type:        N/A
Child Elements:   none

Attribute:        SiteGUID
Description:      GUID that uniquely identifies the site.
DataType:         uuid
Use:              required
```

1.17.8 Examples

This XML sample registers a Site:

1.18 Listener Messages

1.18.1 Overview

Once an ET Provider has successfully registered the set of products (and the highest Version numbers it knows about) it will be reporting on and once the Site has been registered and put online, it can begin sending dynamic update information that includes the following types of information:

- Instance Node declarations (new Instance Nodes), and Instance Node deletions
- Instance Node Property updates
- Alarm Attribute updates (in the form of Alarm Events)
- Simple Events

This type of information represents the bulk of the dynamic information that the ET Provider is responsible for providing to the Enterprise Monitoring Site.

This section describes the XML syntax for delivering these four types of message data.

1.18.2 Instance Node Declarations

New Instance Node declarations can be supplied by sending a <DeclareInstanceNode> message within a <ListenerMessages> XML document via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListener/ETListener.dll?ListenerMessages
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to declare new Instance Nodes:

```
<ListenerMessages ETVersion='3.0' SiteGUID='guid'
ProductNamespaceGUID='guid'>
  <DeclareInstanceNode OID='oid' IID='IID in the form 12/Foo/Logger
A/piml'>
    <Name>Node name</Name>
    <Property Name='name' Value='value' DataType='datatype' Time='UTC
time' />
    <Property Name='name' Value='value' DataType='datatype' Time='UTC
time' />
    <!-- ...  (not all Property values need to be sent when the
Instance Node is created.-->
  </DeclareInstanceNode>
</ListenerMessages>
```

A <ListenerMessages> document can contain an arbitrary number of <DeclareInstanceNode> elements.

As described in section 1.11 and 1.12.1, when an Instance Node is declared and its IID supplied, it is important that all of its parent Instance Nodes have been already declared. If this has not been done, a generic kind of parent Instance Node will be implicitly created which is not necessarily what you want. If this is done, a warning error is returned.

The Listener's response to a <ListenerMessages> document is in the following form:

```
<ListenerMessagesResponse ETVersion='3.0' Status='HRESULT'>
Error text if HRESULT is non zero
  <DeclareInstanceNode IID='IID in the form 12/Foo/Logger A/plm1'
Status='HRESULT'>
  Error text if HRESULT is non zero
  </DeclareInstanceNode>
</ListenerMessagesResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the Status value for the <ListenerMessagesResponse> element is zero, it means that all messages in the document were processed successfully. In this case, no child elements will be contained in this document.

If either the SiteGUID or ProductNamespaceGUID are unknown, the MSGMGR_E_KEYDOESNOTEXIST will be returned as the Status value of the <ListenerMessages> element. In this case, no child elements will be contained in this document.

In all other cases, if an error occurred in the processing of one or more of the child elements, the Status value for the <ListenerMessages> element will be ET_E_ERRORSCONTAINED and the document will contain a child element and Status value for each of the child elements of the original document. (This is different than some of the other document types where a single error will result in the rejection of the entire set of messages.) At least one of these child elements will contain a non-zero Status value which should be processed by the calling software.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[SiteGUID]|[ProductNamespaceGUID]|[OID]|[PropertyName]|[AlarmAttributeName]' does not exist.
0x0022000B (228235)	MSGMGR_S_UNDECLAREDINSTANCENODES	Undeclared parent nodes. The instance nodes %1 for this node have not been defined. Generic instance nodes have been created.

As described in section 1.12.1, when you declare a new Instance Node, you must have previously declared all of its parent nodes first. If this is not done, you will receive the MSGMGR_S_UNDECLAREDINSTANCENODES success code. (Note that the severity bit 31 is not set for this Status value.) In this case, the Instance Node is created (thus, the success code), but because at least one of its parent nodes was not previously declared, a generic Instance Node of type ET_UnknownNodeType is created for the undeclared parent Instance Node(s). If this happens, you can fix the Instance Node type by issuing a <FixInstanceNode> message at a later time.

1.18.3 Fixing Instance Nodes

You can modify the OID type of Instance Nodes implicitly created with type ET_UnknownNodeType during a <DeclareInstanceNode> message. To do this, you send a <FixInstanceNode> message within a <ListenerMessages> XML document. This can only be done to Instance Nodes which are of type ET_UnknownNodeType. This message will be rejected for Instance Nodes of any other type.

To send a <FixInstanceNode> message, you include it in a <ListenerMessages> XML document and send it via the HTTP POST method to the following URN:

`https://ListenerURL/ETListener/ETListener.dll?ListenerMessages`

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to declare fix Instance Nodes:

```
<ListenerMessages ETVersion='3.0' SiteGUID='guid'
ProductNamespaceGUID='guid'>
    <FixInstanceNode OID='oid' IID='IID in the form 12/Foo/Logger
A/pim1'>
        <Name>Node name</Name>
    </FixInstanceNode>
</ListenerMessages>
```

A <ListenerMessages> document can contain an arbitrary number of <FixInstanceNode> elements along with other supported message types.

The OID attribute passed must be the new OID for the Instance Node specified by the IID attribute. You can optionally specify a new Instance Node name via the <Name> tag, if necessary.

The Listener's response to a <ListenerMessages> document is in the following form:

```
<ListenerMessagesResponse ETVersion='3.0' Status='HRESULT'>
Error text if HRESULT is non zero
  <FixInstanceNode IID='IID in the form 12/Foo/Logger A/pim1'
Status='HRESULT'>
    Error text if HRESULT is non zero
  </FixInstanceNode>
</ListenerMessagesResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the Status value for the <ListenerMessagesResponse> element is zero, it means that all messages in the document were processed successfully. In this case, no child elements will be contained in this document.

If either the SiteGUID or ProductNamespaceGUID are unknown, the MSGMGR_E_KEYDOESNOTEXIST will be returned as the Status value of the <ListenerMessages> element. In this case, no child elements will be contained in this document.

In all other cases, if an error occurred in the processing of one or more of the child elements, the Status value for the <ListenerMessages> element will be ET_E_ERRORSCONTAINED and the document will contain a child element and Status value for each of the child elements of the original document. (This is different than some of the other document types where a single error will result in the rejection of the entire set of messages.) At least one of these child elements will contain a non-zero Status value which should be processed by the calling software.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[SiteGUID]|[ProductNamespaceGUID]|[OID]|[IID]' does not exist.

0x8022000C (-2145255413)	MSGMGR_E_INVALIDNODETYPE	The required Instance Node OID is not the correct type for this operation.

MSGMR_E_INVALIDNODETYPE may be returned if the target Instance Node is not of type ET_UnknownNodeType.

1.18.4 Instance Node Deletions

An Instance Node can be deleted by sending a <DeleteInstanceNode> message. You should be very careful with this type of message since it has many implications. When an Instance Node is deleted, all of its child nodes are also deleted. In addition, all Alarm objects associated with the Instance Node and its children are deleted as well.

To send a <DeleteInstanceNode> message, you include it in a <ListenerMessages> XML document and send it via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to delete an Instance Node:

```
<ListenerMessages ETVersion='3.0' SiteGUID='guid'
ProductNamespaceGUID='guid'>
  <DeleteInstanceNode IID='IID in the form 12/Foo/Logger A/pim1' />
</ListenerMessages>
```

A <ListenerMessages> document can contain an arbitrary number of <DeleteInstanceNode> elements along with other supported message types.

The Listener's response to a <ListenerMessages> document is in the following form:

```
<ListenerMessagesResponse ETVersion='3.0' Status='HRESULT'>
Error text if HRESULT is non zero
  <DeleteInstanceNode IID='IID in the form 12/Foo/Logger A/pim1'
Status='HRESULT'>
   Error text if HRESULT is non zero
  </DeleteInstanceNode>
</ListenerMessagesResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the Status value for the <ListenerMessagesResponse> element is zero, it means that all messages in the document were processed successfully. In this case, no child elements will be contained in this document.

If either the SiteGUID or ProductNamespaceGUID are unknown, the MSGMGR_E_KEYDOESNOTEXIST will be returned as the Status value of the

<ListenerMessages> element. In this case, no child elements will be contained in this document.

In all other cases, if an error occurred in the processing of one or more of the child elements, the Status value for the <ListenerMessages> element will be ET_E_ERRORSCONTAINED and the document will contain a child element and Status value for each of the child elements of the original document. (This is different than some of the other document types where a single error will result in the rejection of the entire set of messages.) At least one of these child elements will contain a non-zero Status value which should be processed by the calling software.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[SiteGUID]|[ProductNamespaceGUID]|[IID]' does not exist.

1.18.5 Property Updates

You can send Instance Node Property value updates with a <ListenerEvent> message.

To send a <ListenerEvent> message, you include it in a <ListenerMessages> XML document and send it via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to update a Property of an InstanceNode:

```
<ListenerMessages ETVersion='3.0' SiteGUID='guid'
ProductNamespaceGUID='guid'>
   <ListenerEvent IID='IID in the form 12/Foo/Logger A/pim1' OID='oid'>
      <Property Name="name" Value="value" DataType='datatype' Time="UTC
time" />
   </ListenerEvent>
</ListenerMessages>
```

A <ListenerMessages> document can contain an arbitrary number of <ListenerEvent> elements along with other supported message types.

A <ListenerEvent> element that contains a <Property> child element can contain exactly one <Property> element. To update more than one Property, send additional <ListenerEvent> messages.

The Listener's response to a <ListenerMessages> document is in the following form:

```
<ListenerMessagesResponse ETVersion='3.0' Status='HRESULT'>
Error text if HRESULT is non zero
  <ListenerEvent IID='IID in the form 12/Foo/Logger A/pim1'
Status='HRESULT'>
    Error text if HRESULT is non zero
  </ListenerEvent>
</ListenerMessagesResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the Status value for the <ListenerMessagesResponse> element is zero, it means that all messages in the document were processed successfully. In this case, no child elements will be contained in this document.

If either the SiteGUID or ProductNamespaceGUID are unknown, the MSGMGR_E_KEYDOESNOTEXIST will be returned as the Status value of the <ListenerMessages> element. In this case, no child elements will be contained in this document.

In all other cases, if an error occurred in the processing of one or more of the child elements, the Status value for the <ListenerMessages> element will be ET_E_ERRORSCONTAINED and the document will contain a child element and Status value for each of the child elements of the original document. (This is different than some of the other document types where a single error will result in the rejection of the entire set of messages.) At least one of these child elements will contain a non-zero Status value which should be processed by the calling software.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[SiteGUID]|[ProductNamespaceGUID]|[IID]|[PropertyName]' does not exist.

0x8022000D (-2145255412)	MSGMGR_E_INVALIDVALUETYPE	The data value passed could not be coerced to the proper type.

1.18.6 Alarm Events (Alarm Attribute Updates)

You can send Instance Node Alarm Attribute updates with a <ListenerEvent> message.

To send a <ListenerEvent> message, you include it in a <ListenerMessages> XML document and send it via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to send an Alarm Attribute update for an InstanceNode:

```
<ListenerMessages ETVersion='3.0' SiteGUID='guid'
ProductNamespaceGUID='guid'>

   <ListenerEvent IID='IID in the form 12/Foo/Logger A/pim1' OID='oid'>
      <AlarmAttribute Name="Alarm Attribute name" State='Raise or Clear'
Time="UTC time">
         <MsgID Value="32-bit unsigned integer value">
            <Arg>Arg1</Arg>
            <Arg>Arg2</Arg>
            <!-- . . up to 5 -->
         </MsgID>
      </AlarmAttribute>
   </ListenerEvent>

</ListenerMessages>
```

A <ListenerMessages> document can contain an arbitrary number of <ListenerEvent> elements along with other supported message types.

A <ListenerEvent> element that contains an <AlarmAttribute> child element can contain exactly one <AlarmAttribute> element. To update more than one Alarm Attribute, send additional <ListenerEvent> messages.

As described in section 1.9.1, it is not necessary to use a MsgID Value with substitution arguments. If this is the case, you can send a MsgID Value='0' and include the full message text as a single child <Arg> element.

The Listener's response to a <ListenerMessages> document is in the following form:

```
<ListenerMessagesResponse ETVersion='3.0' Status='HRESULT'>
Error text if HRESULT is non zero
   <ListenerEvent IID='IID in the form 12/Foo/Logger A/pim1'
Status='HRESULT'>
      Error text if HRESULT is non zero
```

```
    </ListenerEvent>
</ListenerMessagesResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the Status value for the <ListenerMessagesResponse> element is zero, it means that all messages in the document were processed successfully. In this case, no child elements will be contained in this document.

If either the SiteGUID or ProductNamespaceGUID are unknown, the MSGMGR_E_KEYDOESNOTEXIST will be returned as the Status value of the <ListenerMessages> element. In this case, no child elements will be contained in this document.

In all other cases, if an error occurred in the processing of one or more of the child elements, the Status value for the <ListenerMessages> element will be ET_E_ERRORSCONTAINED and the document will contain a child element and Status value for each of the child elements of the original document. (This is different than some of the other document types where a single error will result in the rejection of the entire set of messages.) At least one of these child elements will contain a non-zero Status value which should be processed by the calling software.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[SiteGUID]|[ProductNamespaceGUID]|[IID]|[AlarmAttributeName]' does not exist.

1.18.7 Simple Events

You can send Simple Events with a <ListenerEvent> message.

To send a <ListenerEvent> message, you include it in a <ListenerMessages> XML document and send it via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListener/ETListener.dll?ListenerMessages
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to send a Simple Event for an InstanceNode:

```
<ListenerMessages ETVersion='3.0' SiteGUID='guid'
ProductNamespaceGUID='guid'>
    <ListenerEvent IID='IID in the form 12/Foo/Logger_A/pim1' OID='oid'>
      <SimpleEvent Time="UTC time">
        <MsgID Value="32-bit unsigned integer value">
          <Arg>Arg1</Arg>
          <Arg>Arg2</Arg>
          <!-- ... up to 5-->
        </MsgID>
      </SimpleEvent>
    </ListenerEvent>
</ListenerMessages>
```

A <ListenerMessages> document can contain an arbitrary number of <ListenerEvent> elements along with other supported message types.

A <ListenerEvent> element that contains an <SimpleEvent> child element can contain exactly one <SimpleEvent> element. To send than one Simple Event, send additional <ListenerEvent> messages.

As described in section 1.9.1, it is not necessary to use a MsgID Value with substitution arguments. If this is the case, you can send a MsgID Value='0' and include the full message text as a single child <Arg> element.

The Listener's response to a <ListenerMessages> document like this is in the following form:

```
<ListenerMessagesResponse ETVersion='3.0' Status='HRESULT'>
Error text if HRESULT is non zero
    <ListenerEvent IID='IID in the form 12/Foo/Logger_A/pim1'
Status='HRESULT'>
      Error text if HRESULT is non zero
    </ListenerEvent>
</ListenerMessagesResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the Status value for the <ListenerMessagesResponse> element is zero, it means that all messages in the document were processed successfully. In this case, no child elements will be contained in this document.

If either the SiteGUID or ProductNamespaceGUID are unknown, the MSGMGR_E_KEYDOESNOTEXIST will be returned as the Status value of the

<ListenerMessages> element. In this case, no child elements will be contained in this document.

In all other cases, if an error occurred in the processing of one or more of the child elements, the Status value for the <ListenerMessages> element will be ET_E_ERRORSCONTAINED and the document will contain a child element and Status value for each of the child elements of the original document. (This is different than some of the other document types where a single error will result in the rejection of the entire set of messages.) At least one of these child elements will contain a non-zero Status value which should be processed by the calling software.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[SiteGUID]|[ProductNamespaceGUID]|[IID]' does not exist.

1.18.8 Heartbeat Messages

You can send Heartbeat messages with a <HeartbeatMessage> message.

To send a <HeartbeatMessage> message, you include it in a <ListenerMessages> XML document and send it via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

Since Heartbeat messages are not associated with any particular product namespace, this message must be sent as a single message in a <ListenerMessages> XML document with a special administrative ProductNamespace='11111111-1111-1111-1111-111111111111'.

The following XML template is used to send a Heartbeat message to a specific Listener:

```
<ListenerMessages ETVersion='3.0' SiteGUID='guid'
ProductNamespaceGUID='11111111-1111-1111-1111-111111111111'>
  <HeartbeatMessage Source='source name' Destination='A or B' />
</ListenerMessages>
```

This special type of <ListenerMessages> document should contain only a single <HeartbeatMessage> element.

Unlike other messages which can be sent to either Listener A or Listener B, a Heartbeat message must only be sent to the Destination specified by the Destination attribute in the message.

The Listener's response to a <ListenerMessages> document like this is in the following form:

```
<ListenerMessagesResponse ETVersion='3.0' Status='HRESULT'>
Error text if HRESULT is non zero
  <HeartbeatMessage Status='HRESULT'>
    Error text if HRESULT is non zero
  </HeartbeatMessage>
</ListenerMessagesResponse>
```

The Status attribute value is a 32-bit integer. A value of zero indicates that the message was processed successfully. A negative value indicates an error occurred. If the value is non-zero, the element's text will contain an error message indicating why the message could not be processed successfully.

If the Status value for the <ListenerMessagesResponse> element is zero, it means that all messages in the document were processed successfully. In this case, no child elements will be contained in this document.

If the SiteGUID is unknown, the MSGMGR_E_KEYDOESNOTEXIST will be returned as the Status value of the <ListenerMessages> element. In this case, no child elements will be contained in this document.

If the ProductNamespaceGUID is not '11111111-1111-1111-1111-111111111111', MSGMGR_E_INVALIDPRODUCTNAMESPACE will be returned as the Status value of the <ListenerMessages> element. In this case, no child elements will be contained in this document.

In all other cases, if an error occurred in the processing of the <HeartbeatMessage> element, the Status value for the <ListenerMessages> element will be ET_E_ERRORSCONTAINED and the document will contain a <HeartbeatMessage> child element and Status value. The Status value of the <HeartbeatMessage> will contain the specific error value.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	'[SiteGUID]|[ProductNamespaceGUID]|[IID]' does not exist.
0x8022000E (-2145255410)	MSGMGR_E_INVALIDPRODUCTNAMESPACE	The ProductNamespaceGUID specified is invalid in this context.
0x8022000F (-2145255409)	MSGMGR_E_INVALIDHEARTBEATSOURCE	The Source specified in the Heartbeat message does not match what was sent in the SiteOnline message sent earlier.

1.18.9 XML Schemas

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:tns='urn:www.cisco.com:ListenerMessages'
targetNamespace='urn:www.cisco.com:ListenerMessages'>

<xsd:element name='ListenerMessages'/>
  <xsd:complexType>
    <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
      <xsd:element name='ListenerEvent' minOccurs='0'
maxOccurs='unbounded'>
        <xsd:complexType>
          <xsd:choice>
            <xsd:element name='Property'>
              <xsd:complexType>
                <xsd:attribute name='Name' type='xsd:string'
use='required' />
                <xsd:attribute name='Value' type='xsd:string'
use='required' />
                <xsd:attribute name='DataType'
type='xsd:unsignedShort' use='required' />
                <xsd:attribute name='Time' type='xsd:dateTime'
use='required' />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name='AlarmAttribute'>
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name='MsgID'>
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name='Arg'
minOccurs='0' maxOccurs='5' />
                      </xsd:sequence>
                      <xsd:attribute name='Value'
type='xsd:unsignedInt' use='required' />
                    </xsd:complexType>
```

```
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name='Name' type='xsd:string'
use='required' />
                <xsd:attribute name='State' type='tns:AlarmState'
use='required' />
                <xsd:attribute name='Time' type='xsd:dateTime'
use='required' />
            </xsd:complexType>
        </xsd:element>
        <xsd:element name='SimpleEvent'>
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name='MsgID'>
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name='Arg'
minOccurs='0' maxOccurs='5' />
                            </xsd:sequence>
                            <xsd:attribute name='Value'
type='xsd:unsignedInt' use='required' />
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name='Time' type='xsd:dateTime'
use='required' />
            </xsd:complexType>
        </xsd:element>
    </xsd:choice>
    <xsd:attribute name='IID' type='xsd:string' use='required'
/>
    <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required' />
  </xsd:complexType>
 </xsd:element>
 <xsd:element name='DeclareInstanceNode' minOccurs='0'
maxOccurs='unbounded'>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name='Name' type='xsd:string' />
    <xsd:element name='Property' minOccurs='0'
maxOccurs='unbounded'>
     <xsd:complexType>
      <xsd:attribute name='Name' type='xsd:string'
use='required' />
      <xsd:attribute name='Value' type='xsd:string'
use='required' />
      <xsd:attribute name='DataType'
type='xsd:unsignedShort' use='required' />
      <xsd:attribute name='Time' type='xsd:dateTime'
use='required' />
     </xsd:complexType>
```

```
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name='IID' type='xsd:string' use='required'
/>
            <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required' />
        </xsd:complexType>
    </xsd:element>
    <xsd:element name='FixInstanceNode' minOccurs='0'
maxOccurs='unbounded'>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name='Name' type='xsd:string' minOccurs='0'
/>
            </xsd:sequence>
            <xsd:attribute name='IID' type='xsd:string' use='required'
/>
            <xsd:attribute name='OID' type='xsd:unsignedInt'
use='required' />
        </xsd:complexType>
    </xsd:element>
    <xsd:element name='DeleteInstanceNode' minOccurs='0'
maxOccurs='unbounded'>
        <xsd:complexType>
            <xsd:attribute name='IID' type='xsd:string' use='required'
/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name='HeartbeatMessage' minOccurs='0'>
        <xsd:complexType>
            <xsd:attribute name='Source' type='xsd:string'
use='required' />
            <xsd:attribute name='Destination' type='tns:ETSide'
use='required' />
        </xsd:complexType>
    </xsd:element>

  </xsd:sequence>
  <xsd:attribute name='ETVersion' type='xsd:float' use='required' />
  <xsd:attribute name='SiteGUID' type='tns:GUID' use='required' />
  <xsd:attribute name='ProductNamespaceGUID' type='tns:GUID'
use='required' />
 </xsd:complexType>
</xsd:element>

<xsd:simpleType name='GUID'>
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value='[0-9A-Fa-f]{8}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{4}-
[0-9A-Fa-f]{4}-[0-9A-Fa-f]{12}' />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name='AlarmState'>
  <xsd:restriction base='xsd:string'>
```

```
      <xsd:enumeration value='Raise' />
      <xsd:enumeration value='Clear' />
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name='ETSide'>
    <xsd:restriction base='xsd:string'>
      <xsd:enumeration value='A' />
      <xsd:enumeration value='B' />
    </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

1.18.10XML Tag Descriptions

The following XML tags are defined in the <ListenerMessages> document:

```
Tag:                <ListenerMessages> (root element)
Description:        Defines the root element for the ListenerMessages
document. This type of document contains dynamic updates to the set
of Instance Nodes being monitored at the Enterprise Monitoring Site.
Content Type:       elementOnly
Data Type:          N/A
Child Elements:     <ListenerEvent> [0,…]
                    <DeclareInstanceNode> [0,…]
                    <FixInstanceNode> [0,…]
                    <DeleteInstanceNode> [0,…]
                    <HeartbeatMessage> [0,1]

Attribute:          ETVersion
Description:        ET protocol version number of document. Must be 3.0.
DataType:           float
Use:                required
Value:              3.0

Attribute:          SiteGUID
Description:        GUID that uniquely identifies the site which is the
source of this set of messages.
DataType:           uuid
Use:                required

Attribute:          ProductNamespaceGUID
Description:        GUID that uniquely identifies the product.
DataType:           uuid
Use:                required


Tag:                <ListenerEvent>
Description:        A message containing a dynamic update to an existing
Instance Node. This message must contain exactly one of a Property
update, an Alarm Event or a Simple Event.
Content Type:       elementOnly
Data Type:          N/A
```

Child Elements:	<Property> [0,1] <AlarmAttribute> [0,1] <SimpleEvent> [0,1]
Attribute:	**IID**
Description:	Instance Identifier string that uniquely defines the Instance Node within the product namespace and site. The IID must be described in a hierarchical fashion using the syntax described in section 1.11.
DataType:	string
Use:	required
Attribute:	**OID**
Description:	Unique (within a product namespace) Object Identifier that defines the type of Instance Node referred to by the IID. (Note that this value should already be known from the <DeclareInstanceNode> message, but it is used as an optimization hint for the server software, and as a check mechanism in case the Instance Node was implicitly created with an ET_UnknownNodeType class type.
DataType:	unsignedInt
Use:	required

Tag:	**<Property>**
Description:	Contains an update to a Property value.
Content Type:	empty
Data Type:	N/A
Child Elements:	none
Attribute:	**Name**
Description:	The Property name being updated.
DataType:	string
Use:	required
Attribute:	**Value**
Description:	The new value of the Property.
DataType:	string
Use:	required
Attribute:	**DataType**
Description:	The data type of the Property. The value should be one of the supported VARENUM values as described in section 1.8.4.
DataType:	unsignedShort
Use:	required
Attribute:	**Time**
Description:	The date and time the Property changed value. This time must be in UTC time (GMT/Zulu) using the dateTime format as defined by the W3C XSD specification. See the Examples section (section 1.18.11) for a sample of the dateTime field.
DataType:	dateTime
Use:	required

Tag:	**<AlarmAttribute>**

Description: Contains an Alarm object Event. This updates the state of an Alarm Attribute.
Content Type: elementOnly
Data Type: N/A
Child Elements: <MsgID> [1,1]

Attribute: Name
Description: The Property name being updated.
DataType: string
Use: required

Attribute: State
Description: The new state of the Alarm Attribute. This must be either the word 'Raise' or the word 'Clear'.
DataType: string
Use: required

Attribute: Time
Description: The date and time of the event. This time must be in UTC time (GMT/Zulu) using the dateTime format as defined by the W3C XSD specification. See the Examples section (section 1.18.11) for a sample of the dateTime field.
DataType: dateTime
Use: required

Tag: <MsgID>
Description: Defines the MsgID associated with this Event. The MsgID cross references a previously defined static message text field. See section 1.9 for a description of how MsgIDs are used, and section 1.15 for information on how to supply the static message text fields.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Arg> [0,5]

Attribute: Value
Description: Message ID associated with the Event.
DataType: unsignedInt
Use: required

Tag: <Arg>
Description: Text strings (from zero (0) to five (5)) containing substituion arguments for the MsgID message text field. See section 1.9 for a description of how substitution arguments are used with MsgIDs. Note that if a MsgID Value of zero is used, it is assumed that only one <Arg> element is included, and that element must contain the full text of the message. No substitutions are made in this case.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: <SimpleEvent>
Description: Contains a Simple Event.
Content Type: elementOnly

Data Type: N/A
Child Elements: <MsgID> [1,1]

Attribute: Time
Description: The date and time of the event. This time must be in UTC time (GMT/Zulu) using the dateTime format as defined by the W3C XSD specification. See the Examples section (section 1.18.11) for a sample of the dateTime field.
DataType: dateTime
Use: required

Tag: <DeclareInstanceNode>
Description: Declares a new Instance Node. For the IID specified, it is important that all of its parent Instance Nodes have been previously defined.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Name> [1,1]
<Property> [0,*]

Attribute: IID
Description: Instance Identifier string that uniquely defines the Instance Node within the product namespace and site. The IID must be described in a hierarchical fashion using the syntax described in section 1.11.
DataType: string
Use: required

Attribute: OID
Description: Unique (within a product namespace) Object Identifier that defines the type of Instance Node referred to by the IID.
DataType: unsignedInt

Tag: <Name>
Description: The display name of the Instance Node.
Content Type: textOnly
Data Type: string
Child Elements: none

Tag: <FixInstanceNode>
Description: Fixes the OID value for an Instance Node implicitly created with a <DeclareInstanceNode> message. This message can only be used for an existing Instance Node which is of class type ET_UnknownNodeType.
Content Type: elementOnly
Data Type: N/A
Child Elements: <Name> [0,1]

Attribute: IID
Description: Instance Identifier string that uniquely defines the Instance Node within the product namespace and site. The IID must be

described in a hierarchical fashion using the syntax described in section 1.11.
DataType: string
Use: required

Attribute: OID
Description: New Object Identifier that defines the type of Instance Node referred to by the IID.
DataType: unsignedInt

Tag: <DeleteInstanceNode>
Description: Deletes an existing Instance Node and all of its child nodes. This message also deletes all Alarm history data associated with this node.
Content Type: empty
Data Type: N/A
Child Elements: none

Attribute: IID
Description: Instance Identifier string that uniquely defines the Instance Node within the product namespace and site to be deleted.
DataType: string
Use: required

Tag: <HeartbeatMessage>
Description: Contains a Heartbeat message to indicate the health of the connection between the ET Provider and Listener. More information on Heartbeat messages can be found in sections 1.12.5and 1.22.
Content Type: empty
Data Type: N/A
Child Elements: none

Attribute: Source
Description: Display name of the source of the Heartbeat message. This name must match what was previously sent in the <ListenerSiteAdmin> <SiteOnline> message described in section 1.17 and 1.17.3.
DataType: string
Use: required

Attribute: Destination
Description: A single letter ('A' or 'B') that indicates which Listener this message was sent to.
DataType: string
Use: required

1.18.11Examples
TBD

1.19 Queries

The ET Provider specification allows you to execute a limited number of queries to get configuration information from the Enterprise Monitoring Site. This query capability is most likely to be used at the Enterprise Monitoring Site to build administrative applications. It is less likely that an ET Provider would need to execute these queries.

The query types fall into the following categories:

- Product Namespace queries
- Class Definition queries
- Product Version delta queries
- MsgID queries
- Description queries
- Site information queries (this type of query is restricted to users in the ETAdmin group)
- Instance Node queries (this type of query is restricted to users in the ETAdmin group)

These query types are described in more detail in the following sections.

1.19.1 Product Namespace Queries

To query an Enterprise Monitoring Site for knowledge of a particular product or a group of products, an ET Provider sends an XML document to Listener with a root tag of <ListenerQuery> and a single <ProductNamespaces> child element. The XML document must be sent via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListener/ETListener.dll?ListenerMessages
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to query an Enterprise Monitoring Site for information about a product:

```
<ListenerQuery ETVersion='3.0'>
    <ProductNamespaces>
        <ProductNamespaceGUID>guid</ProductNamespaceGUID>
    </ProductNamespaces>
</ListenerQuery>
```

A <ListenerQuery> document can contain exactly one child element of which <ProductNamespaces> is one of the allowable types.

The <ProductNamespaceGUID> element is optional. If not present, information on all registered product namespaces will be returned. One or more

<ProductNamespaceGUID> elements can be included. Information on each product namespace will be returned in the response document.

The Listener's response to the above document type is in the form:

```
<ListenerQueryResponse ETVersion='3.0' Status='HRESULT'>
   <ErrorText>error text</ErrorText>
   <ProductNamespaces>
      <ProductNamespace ProductNamespaceGUID='guid'>
         <Name>product name</Name>
         <Manufacturer>product manufacturer</Manufacturer>
         <Description>product description</Description>
         <Versions>
            <Version ModifiedTime='time'>1.0</Version>
            <Version ModifiedTime='time'>1.5</Version>
            <Version ModifiedTime='time'>2.0</Version>
         </Versions>
         <ModifiedTime>time</ModifiedTime>
      </ProductNamespace>
   </ProductNamespaces>
```

The Status attribute associated with the <ListenerQuery> element represents the overall status of processing the document. If the Status value is non-zero, it means that some error occurred in the processing. In this case, the document will contain a single child element called <ErrorText> that contains some information about the error that occurred. Possible Status values are shown in the table below.

If the Status value is zero, then the query was processed successfully and the data returned will be as seen above. In this case, the <ErrorText> element will be absent.

For successful queries, there will be a <ProductNamespace> element for each product namespace requested, or a full set of product namespaces if the <ProductNamespace> element is omitted from the request document.

Note that the <Versions> element contains a list of all of the previously registered Version numbers for the product.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

In addition some database-specific HRESULTs may be returned.

1.19.2 Class Definition Queries

To query an Enterprise Monitoring Site for knowledge of a particular Class Definition or group of Class Definitions for a particular product, an ET Provider sends an XML document to Listener with a root tag of <ListenerQuery> and a single

<ClassDefintions> child element. The XML document must be sent via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to query an Enterprise Monitoring Site for information about Class Definitions:

```
<ListenerQuery ETVersion='3.0'>
   <ClassDefinitions ProductNamespaceGUID='guid'>
      <ClassDefinition OID='oid' />
   </ClassDefinitions>
</ListenerQuery>
```

A <ListenerQuery> document can contain exactly one child element of which <ClassDefinitions> is one of the allowable types.

The <ClassDefinition> element is optional. If not present, information on all registered Class Definitions for the product namespace will be returned. One or more <ClassDefinition> elements can be included. Information on each Class Definition will be returned in the response document.

The Listener's response to the above document type is in the form:

```
<ListenerQueryResponse ETVersion='3.0' Status='HRESULT'>
   <ErrorText>error text</ErrorText>
   <ClassDefinitions ProductNamespaceGUID='guid'>
      <ClassDefinition OID='oid'>
      <!-- format just like ClassDefinition tag in RegisterProduct
message -->
      <ModifiedTime>time</ModifiedTime>
      </ClassDefinition>
   </ClassDefinitions>
</ListenerQueryResponse>
```

The Status attribute associated with the <ListenerQuery> element represents the overall status of processing the document. If the Status value is non-zero, it means that some error occurred in the processing. In this case, the document will contain a single child element called <ErrorText> that contains some information about the error that occurred. Possible Status values are shown in the table below.

If the Status value is zero, then the query was processed successfully and the data returned will be as seen above. In this case, the <ErrorText> element will be absent.

For successful queries, there will be a <ClassDefinition> element for each Class Defintion requested, or a full set of Class Definitions if the <ClassDefinition> element is omitted from the request document.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

In addition some database-specific HRESULTs may be returned.

1.19.3 Product Version Delta Queries

To query an Enterprise Monitoring Site for the XML document that describes the transition of one product version to the next, an ET Provider sends an XML document to Listener with a root tag of <ListenerQuery> and a single <ProductVersionDelta> child element. The XML document must be sent via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListener/ETListener.dll?ListenerMessages
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to query an Enterprise Monitoring Site for information about a product version change:

```
<ListenerQuery ETVersion='3.0'>
    <ProductVersionDelta ProductNamespaceGUID='guid' Version='version
number' />
</ListenerQuery>
```

A <ListenerQuery> document can contain exactly one child element of which <ProductVersionDelta> is one of the allowable types.

The Version attribute defines the NewVersion of the product for which information is required. So for example, if a product has 3 versions defined: 1.0, 1.5 and 2.0; if you set Version='1.5' you are requesting the delta document that describes the transition of the Class Definitions from Version 1.0 to Version 1.5 of the product.

The Listener's response to the above document type is in the form:

```
<ListenerQueryResponse ETVersion='3.0' Status='HRESULT'>
    <ErrorText>error text</ErrorText>
    <ProductVersionDelta ProductNamespaceGUID='guid' Version='version
number'>
        <ClassDefinition OID='oid'>
        <!-- format just like ClassDefinition tag in RegisterProduct
message -->
        </ClassDefinition>
        <ModifiedTime>time</ModifiedTime>
    </ProductVersionDelta>
```

```
</ListenerQueryResponse>
```

The Status attribute associated with the <ListenerQuery> element represents the overall status of processing the document. If the Status value is non-zero, it means that some error occurred in the processing. In this case, the document will contain a single child element called <ErrorText> that contains some information about the error that occurred. Possible Status values are shown in the table below.

If the Status value is zero, then the query was processed successfully and the data returned will be as seen above. In this case, the <ErrorText> element will be absent.

For successful queries, the response should be a document that looks exactly like the one sent by the ET Provider in the <RegisterProduct> message in the <ListenerProductAdmin> document as described in section 1.14.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

In addition some database-specific HRESULTs may be returned.

1.19.4 MsgID Queries

To query an Enterprise Monitoring Site for knowledge of a particular MsgID or a set of MsgIDs, an ET Provider sends an XML document to Listener with a root tag of <ListenerQuery> and a single <MsgIDs> child element. The XML document must be sent via the HTTP POST method to the following URN:

```
https://ListenerURL/ETListener/ETListener.dll?ListenerMessages
```

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to query an Enterprise Monitoring Site for information about MsgIDs:

```
<ListenerQuery ETVersion='3.0'>
    <MsgIDs ProductNamespaceGUID='guid' Source='ETProvider, Local,
Current' LanguageID='id'>
        <MsgID>message id</MsgID>
    </MsgIDs>
</ListenerQuery>
```

A <ListenerQuery> document can contain exactly one child element of which <MsgIDs> is one of the allowable types.

The <MsgID> element is optional. If not present, information on all MsgIDs that satisfy the criteria specified by the ProductNamespaceGUID, Source and LanguageID attributes

will be returned. One or more <MsgID> elements can be included. Information on each MsgID will be returned in the response document.

The ProductNamespaceGUID is a required attribute which selects the specific product namespace to query.

The Source attribute is required and can be one of the following three values:

- ETProvider – this returns MsgID descriptions that were defined by the ET Provider
- Local – this returns the MsgID descriptions that were defined by a local override at the Enterprise Monitoring Site
- Current – this returns the MsgID descriptions that are currently in use. The information returned may have been set by the ET Provider or by a local override.

The LanguageID attribute is optional. If present it returns MsgID information only for the language specified. If it is omitted, MsgID information is returned for all known Language Identifiers.

The Listener's response to the above document type is in the form:

```
<ListenerQueryResponse ETVersion='3.0' Status='HRESULT'>
   <ErrorText>error text</ErrorText>
   <MsgIDs ProductNamespaceGUID='guid' Source='ETProvider, Local,
Current'>
      <MsgID Value='32-bit unsigned integer value'>
         <Language ID='id'>
            <MsgText>message text with substitution arguments</MsgText>
            <Description>description of what the Event
means</Description>
            <Action>information on what a support organization should do
when this Event occurs</Action>
            <ModifiedTime>time</ModifiedTime>
         </Language>
      </MsgID>
   </MsgIDs>
</ListenerQueryResponse>
```

The Status attribute associated with the <ListenerQuery> element represents the overall status of processing the document. If the Status value is non-zero, it means that some error occurred in the processing. In this case, the document will contain a single child element called <ErrorText> that contains some information about the error that occurred. Possible Status values are shown in the table below.

If the Status value is zero, then the query was processed successfully and the data returned will be as seen above. In this case, the <ErrorText> element will be absent.

For successful queries, there will be a <MsgID> element for each MsgID requested, or a full set of MsgIDs if the <MsgID> element is omitted from the request document.

If the LanguageID attribute was present in the request document, the only <Language> element returned will be for the one that was requested.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

In addition some database-specific HRESULTs may be returned.

1.19.5 Description Queries

To query an Enterprise Monitoring Site for knowledge of a particular Description or a set of Descriptions, an ET Provider sends an XML document to Listener with a root tag of <ListenerQuery> and a single <Descriptions> child element. The XML document must be sent via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerMessages

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to query an Enterprise Monitoring Site for information about Descriptions:

```
<ListenerQuery ETVersion='3.0'>
    <Descriptions ProductNamespaceGUID='guid' Source='ETProvider, Local,
Current' LanguageID='id'>
        <ProductDescription/>
        <ClassDescription OID='oid' />
        <PropertyDescription OID='oid' Name='property name' />
        <AlarmAttributeDescription OID='oid' Name='Alarm Attribute name'
/>
    </Descriptions>
</ListenerQuery>
```

A <ListenerQuery> document can contain exactly one child element of which <Descriptions> is one of the allowable types.

The <ProductDescription>, <ClassDescription>, <PropertyDescription> and <AlarmAttributeDescription> elements are optional. If not present, information on all Descriptions that satisfy the criteria specified by the ProductNamespaceGUID, Source and LanguageID attributes will be returned. One or more <ClassDescription>, <PropertyDescription> and <AlarmAttributeDescription> elements can be included. Information on each Description will be returned in the response document.

The ProductNamespaceGUID is a required attribute which selects the specific product namespace to query.

The Source attribute is required and can be one of the following three values:

- ETProvider – this returns Descriptions that were defined by the ET Provider
- Local – this returns the Descriptions that were defined by a local override at the Enterprise Monitoring Site
- Current – this returns the Descriptions that are currently in use. The information returned may have been set by the ET Provider or by a local override.

The LanguageID attribute is optional. If present it returns Description information only for the language specified. If it is omitted, Description information is returned for all known Language Identifiers.

The Listener's response to the above document type is in the form:

```
<ListenerQueryResponse ETVersion='3.0' Status='HRESULT'>
  <ErrorText>error text</ErrorText>

  <Descriptions ProductNamespaceGUID='guid' Source='ETProvider, Local,
Current' LanguageID='id'>
    <ProductDescription>
      <Language ID='id'>
        <Description>description of the product</Description>
        <ModifiedTime>time</ModifiedTime>
      </Language>
    </ProductDescription>

    <ClassDescription OID='oid'>
      <Language ID='id'>
        <Description>description of the class</Description>
        <ModifiedTime>time</ModifiedTime>
      </Language>
    </ClassDescription>

    <PropertyDescription OID='oid' Name='property name'>
      <Language ID='id'>
        <Description>description of the Property</Description>
        <ModifiedTime>time</ModifiedTime>
      </Language>
    </PropertyDescription>

    <AlarmAttributeDescription OID='oid' Name='Alarm Attribute name'>
      <Language ID='id'>
        <Description>description of the Alarm Attribute</Description>
        <ModifiedTime>time</ModifiedTime>
      </Language>
    </AlarmAttributeDescription>
  </Descriptions>
</ListenerQueryResponse>
```

The Status attribute associated with the <ListenerQuery> element represents the overall status of processing the document. If the Status value is non-zero, it means that some error occurred in the processing. In this case, the document will contain a single child element called <ErrorText> that contains some information about the error that occurred. Possible Status values are shown in the table below.

If the Status value is zero, then the query was processed successfully and the data returned will be as seen above. In this case, the <ErrorText> element will be absent.

For successful queries, there will be a <*XXX*Description> element for each Description requested, or a full set of Descriptions if the child elements are omitted from the request document.

If the LanguageID attribute was present in the request document, the only <Language> element returned will be for the one that was requested.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

In addition some database-specific HRESULTs may be returned.

1.19.6 Site Information Queries

To query an Enterprise Monitoring Site for knowledge of a particular site or a group of sites, you send an XML document to Listener with a root tag of <ListenerQuery> and a single <Sites> child element.

To perform this query, the sender must be a member of the ETAdmin users group.

The XML document must be sent via the HTTP POST method to the following URN:

`https://ListenerURL/ETListener/ETListener.dll?ListenerAdmin`

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to query an Enterprise Monitoring Site for information about sites:

```
<ListenerQuery ETVersion="3.0">
   <Sites>
      <SiteGUID>guid</SiteGUID>
   </Sites>
</ListenerQuery>
```

A <ListenerQuery> document can contain exactly one child element of which <Sites> is one of the allowable types.

The <SiteGUID> element is optional. If not present, information on all registered sites will be returned. One or more <SiteGUID> elements can be included. Information on each site will be returned in the response document.

The Listener's response to the above document type is in the form:

```
<ListenerQueryResponse ETVersion='3.0' Status='HRESULT'>
  <ErrorText>error text</ErrorText>

  <Sites>
    <Site SiteGUID='guid'>
      <SiteType>Enterprise or Customer</SiteType>
      <SiteName>name</SiteName>
      <DownStreamSiteGUID>guid</DownStreamSiteGUID>
      <Status>0 for offline, 1 for online</Status>
      <ListenerAURL>URL [valid and required only for Enterprise
sites]</ListenerAURL>
      <ListenerBURL>URL [valid and required only for Enterprise
sites]</ListenerBURL>
      <HeartbeatInterval Source='source name'>interval in minutes
that source will send Heartbeat messages</HeartbeatInterval>
      <ReadOnly>true or false</ReadOnly>
      <Notes>text information to indicate why the site is going
offline</Notes>
      <ModifiedTime>time</ModifiedTime>
    </Site>
  </Sites>

</ListenerQueryResponse>
```

The Status attribute associated with the <ListenerQuery> element represents the overall status of processing the document. If the Status value is non-zero, it means that some error occurred in the processing. In this case, the document will contain a single child element called <ErrorText> that contains some information about the error that occurred. Possible Status values are shown in the table below.

If the Status value is zero, then the query was processed successfully and the data returned will be as seen above. In this case, the <ErrorText> element will be absent.

For successful queries, there will be a <Site> element for each site requested, or a full set of sites if the <SiteGUID> element is omitted from the request document.

The following error values have been defined for this type of document:

Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

In addition some database-specific HRESULTs may be returned.

1.19.7 Instance Node Queries

To query an Enterprise Monitoring Site for knowledge of a particular Instance Node or a group of Instance Nodes, you send an XML document to Listener with a root tag of <ListenerQuery> and a single <InstanceNodes> child element.

To perform this query, the sender must be a member of the ETAdmin users group.

The XML document must be sent via the HTTP POST method to the following URN:

https://ListenerURL/ETListener/ETListener.dll?ListenerAdmin

where ***ListenerURL*** is the URL for one of the two Listener machines, e.g., listenera.cisco.com or listenerb.cisco.com.

The following XML template is used to query an Enterprise Monitoring Site for information about sites:

```
<ListenerQuery ETVersion='3.0'>
   <InstanceNodes ProductNamespaceGUID='guid' SiteGUID='guid' />
</ListenerQuery>
```

A <ListenerQuery> document can contain exactly one child element of which <InstanceNodes> is one of the allowable types.

Both the ProductNamespaceGUID and SiteGUID attributes are required.

The Listener's response to the above document type is in the form:

```
<ListenerQueryResponse ETVersion='3.0' Status='HRESULT'>
   <ErrorText>error text</ErrorText>
   <InstanceNodes ProductNamespaceGUID='guid' SiteGUID='guid'>
      <InstanceNode>
         <OID>oid</OID>
         <IID>iid</IID>
         <Name>name</Name>
         <ModifiedTime>time</ModifiedTime>
      </InstanceNode>
   </InstanceNodes>
</ListenerQueryResponse>
```

The Status attribute associated with the <ListenerQuery> element represents the overall status of processing the document. If the Status value is non-zero, it means that some error occurred in the processing. In this case, the document will contain a single child element called <ErrorText> that contains some information about the error that occurred. Possible Status values are shown in the table below.

If the Status value is zero, then the query was processed successfully and the data returned will be as seen above. In this case, the <ErrorText> element will be absent.

For successful queries, there will be a set of <InstanceNode> elements for the product namespace and site requested.

The following error values have been defined for this type of document:

Status Values		
Value	Mnemonic	Error Text
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.

In addition some database-specific HRESULTs may be returned.

Communicating With a Listener

1.20 The Use of HTTPS

1.20.1 Secure Communication

1.20.2 Authentication Using Client Certificates

1.20.3 HTTP Headers

The following HTTP Headers must be included when sending messages to a Listener:

```
Content-Type: text/xml
```

The **text/xml** indicates the mime type for the message. Since all messages contain XML data, this Content-Type should always be part of the HTTP header. If this is missing, the message will be rejected by the Listener.

```
MessageTimeout: [time in seconds before client application will close the HTTP session]
```

The MessageTimeout custom HTTP header must be sent to indicate how long the ET Provider client application will wait for a complete response before timing out. This information is used by the Listener to determine when to send a response back to the client if the MsgMgr process cannot process the data. The minimum allowable value is 60 seconds. Recommended values are in the range of 120-300 seconds.

Note that you should make sure that your messages are properly serialized. That is, you should not send another message until you have received a response back from Listener or until your client application times out.

1.20.4 User Names and Passwords

As part of the process of registering a site with an Enterprise Monitoring Site, you will be required to obtain a User Name and Password to be used for sending your messages. This information is required to authorize your usage of the repository at the Enterprise Monitoring Site. Invalid User Names or Passwords will result in the message being rejected with a 401 Access Denied HTTP response. Since SSL message encryption is used, the clear text User Name and Password will be encrypted along with the data. When you are issued this User Name and Password, you should take care to protect them in your software so that unauthorized users will not have access to the names you have been issued.

1.21 Error Handling

When you communicate with Listener, your client application will typically receive an XML document in the HTTP Response Body. This document will include details on the success of the operation intended. However, other situations could arise that may result in you not receiving this type of response. The following flow diagram illustrates the possible scenarios:

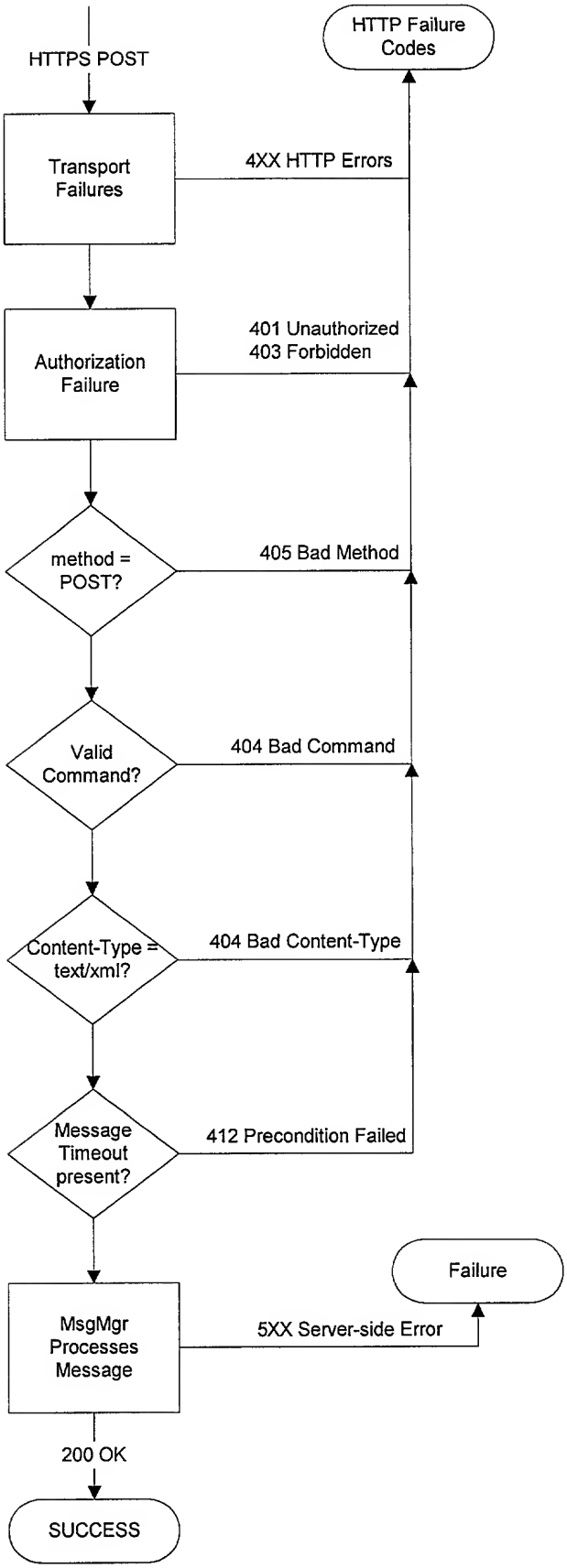
HTTPS POST
HTTP Failure Codes
Transport Failures
4XX HTTP Errors
Authorization Failure
401 Unauthorized
403 Forbidden
method = POST?
405 Bad Method
Valid Command?
404 Bad Command
Content-Type = text/xml?
404 Bad Content-Type
Message Timeout present?
412 Precondition Failed
Failure
MsgMgr Processes Message
5XX Server-side Error
200 OK
SUCCESS

Figure 5 – Error flow for messages sent from ET Provider to Listener.

In the above figure, there are two main classes of errors that can be returned when a document is sent to the Listener:

1. HTTP Request (400-series) HTTP errors
2. Server (500-series) errors.

In addition, an HTTP 200 OK HTTP status may be returned which means the document was successfully processed.

If a 400-series error is returned, the Response Body should contain an XML document of the following form:

```
<HTTPError>
[Text of error here]
</HTTPError>
```

This Response Body differs from the XML documents returned for specific messages because this class of error messages are returned before the Request Body is even parsed.

In extreme cases when the client application cannot contact the Listener server, a mime type of **text/plain** or **text/html** Response Body may be returned.

The 500-series of error messages are returned because the MsgMgr process failed to process all or part of the document. If you receive a 500-series error, you should carefully parse the XML document returned to see what kind of error occurred.

The following specific 500-series Server-side errors have been defined:

HTTP Status	Description	Interpretation
521	Invalid Document Type	An unknown target namespace was specified.
522	Invalid syntax	The XML document was not well formed.
523	Invalid schema	The XML document did not conform to the required XSD sch More details are contained in the XML response document.
524	Logic Error	Some kind of business logic failed in the processing of the messages. More details are contained in the XML response document.
525	Server Timeout	The MsgMgr server was unable to process the document withi specified timeout period.

If you receive a 200 OK HTTP status, you can assume the document was processed in its entirety with no errors. In this case, you should not have to parse the Response Body.

1.21.1 Duplexed Listeners

To provide fault tolerant delivery and processing, the components at the Central Repository Site (Listener and MsgMgr, see Figure 1) are *duplexed*, which means there are duplicate components running on different machines. This gives the system a much higher degree of reliability in both the delivery and processing of incoming messages. A picture of the processing paths at the Central Repository Site is shown in the figure below.

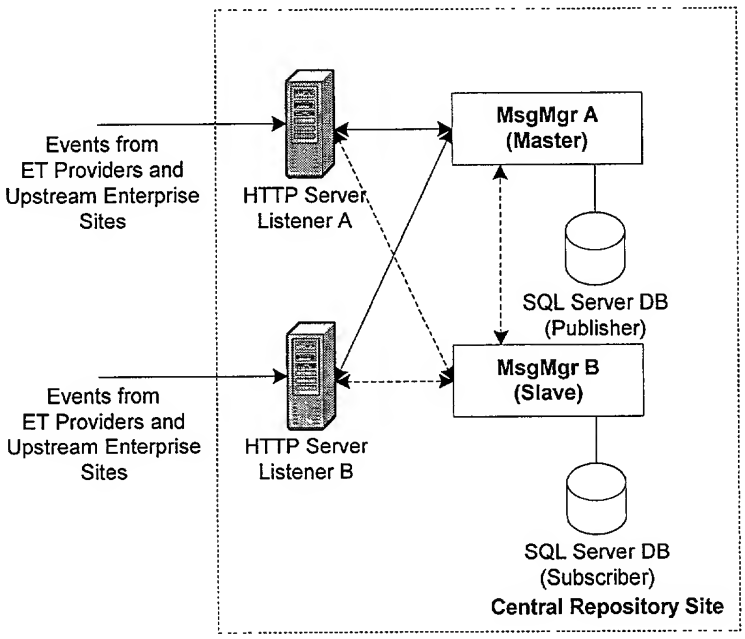

Figure 6 - Data flow at Central Repository Site showing duplexed components.

When your ET Provider sends dynamic updates, you have the option of sending the messages to the URL for Listener A or Listener B. Either one will do. If the message is processed successfully (you receive a 200 OK HTTP response), then you are done with that document. If you receive an error, you will have to figure out what to do next. If the error is a logic error that is your fault (e.g., bad HTTP method, bad HTTP headers, invalid XML schema, business logic errors), then it is up to you to correct these errors and resend the document. If the error is an HTTP transport failure, then it is possible that the Listener you tried to send the document to was unreachable or down. In this case, you can try to resend it to the other Listener. If both are unreachable, then it is up to you to cache the document and resend it. If time order is important (it usually is), you may want to just keep trying both Listeners until the document is successfully delivered before moving on to other documents.

When you build your ET Provider, you have the option of also duplexing the components that send the Event stream to the Listener sites. This is entirely up to you how you manage this.

1.22 Heartbeat Messages

As described in section 1.12.5, Heartbeat messages must be sent by on-line ET Provider Sites to indicate that they are alive and to validate the communication path between the ET Provider and the Listeners. When a Site marks itself as being online by sending a <ListenerSiteAdmin> <SiteOnline> message (see section 1.10, 1.17 and 1.17.3), one of the requirements is to specify at least one <HeartbeatInterval> tag in the <SiteOnline> element. The <HeartbeatInterval> element includes a Source name, which is a display name of your choosing, and the interval in minutes of how often you will send a Heartbeat message to the Listeners. Once your Site is online, it is the ET Provider's responsibility to send this Heartbeat message to the Listeners at the prescribed interval.

For each Source defined in the <SiteOnline> element (there must be at least one, and up to two), it must send a Heartbeat message to both Listener A and Listener B at the prescribed intervals. As described in section 1.12.5, it is recommended (but not required) to send these messages in band using the same mechanism that you use to send other <ListenerMessages> documents. When you send these messages, you must use the special administrative ProductNamespaceGUID='11111111-1111-1111-1111-111111111111' attribute in the <ListenerMessages> root element tag. An example of such a Heartbeat message is shown below:

```
<p:ListenerMessages xmlns:p='urn:www.cisco.com:ListenerMessages'
ETVersion='3.0'
SiteGUID='12345678-1234-1234-1234-1234567890ab'
ProductNamespaceGUID='11111111-1111-1111-1111-111111111111'>
  <HeartbeatMessage Source='DTPA' Destination='B' />
</p:ListenerMessages>
```

Unlike other <ListenerMessages> documents which can be sent to either Listener, Heartbeat messages must be sent only to the Listener marked in the Destination attribute. Thus, when it is time to send a Heartbeat message, the Source would send a Heartbeat message to Listener A only, and then it would send one (like the above example) to Listener B only.

In the example above, the Source (DTPA) must match the Source attribute used in the <SiteOnline> message. For the above message, the <SiteOnline> message might have looked like:

```
<p:ListenerSiteAdmin xmlns:p='urn:www.cisco.com:ListenerSiteAdmin'
ETVersion='3.0'>
  <SiteOnline SiteGUID='12345678-1234-1234-1234-1234567890ab'>
    <HeartbeatInterval Source='DTPA' Interval='15' />
```

```
    <HeartbeatInterval Source='DTPB' Interval='15' />
    <ReadOnly>false</ReadOnly>
  </SiteOnline>
</p:ListenerSiteAdmin>
```

Internationalization

Internationalization is supported in ET in the sense that the message text for Alarm object Events and other Simple Events, and the Description fields for ET objects can be displayed in the desired language of the client application. Support is built into the ET specification to allow ET Providers to supply localized static message text and Description fields that are maintained in the server software's databases. These static text fields are stored for each language supported, and then they can be returned in a client request.

1.23 MsgIDs

It is up to the ET Provider to provide localized versions of the MsgID message text and Descriptions for each Product Namespace supported. For the set of MsgIDs defined for a Product, the ET Provider would provide translations of the MsgID message text and send them to the Enterprise Monitoring Site in a <ListenerProductI18N> XML document. Such a document includes the ProductNamespaceGUID that identifies the product, the MsgID that identifies the message, and a Language Identifier that identifies the language. These three keys will identify the proper message text for a client application.

The details of how to send a set of message descriptions for a set of languages is included in section 1.15.

1.24 Description Fields

In a similar manner, you can also supply localized versions of Description fields used for various ET objects. The ET specification allows you to provide text Description fields associated with the following objects:

- Product
- Class Definition
- Property of a Class Definition
- Alarm Attribute

By convention when you define these objects via the ET protocols described elsewhere in section 1.12.5, you provide <Description> text elements in English (Language Identifier = 1033). The ET specifications allow you to provide these Descriptions in

other languages so that client applications that may want to describe this information can see the text in the language of their choice.

The set of Descriptions must be sent after the product has been registered since the ProductNamespaceGUID must already have been registered. If this is not done, an error results.

Unlike the <ListenerProductAdmin> XML document, the document that includes Descriptions can be sent in total or in bits and pieces. One possible scenario is to send the set of Descriptions for a new language at a different time.

When a set of Descriptions are sent, if the source of the Descriptions is from an ET Provider, a first check is made to see if the ProductVersion being sent in the document matches the ProductVersion that has been registered. If this is not the case, the document will be rejected.

ET Provider Checklist

1.25 Design Strategy

1.25.1 Create a Product Namespace

1.25.2 Create Class Definitions

1.25.3 Satisfy All Versioning Requirements

1.25.4 Create MsgIDs

1.25.5 Supply Additional Language Information as Needed

1.26 Site Registration

1.26.1 Register Your Site

1.26.2 Register All Products

1.26.3 Send MsgIDs and Descriptions

1.27 Dynamic Updates

1.27.1 Bring Your Site Online

1.27.2 Maintain Your Object State

1.27.3 Declare Instance Nodes

1.27.4 Send Dynamic Updates

1.27.5 Send Heartbeats

1.27.6 Putting Your Site Offline

ET Class Definitions

The ET specifications provide a small number of basic class definitions that are part of a global product namespace (ProductNamespaceGUID='00000000-0000-0000-0000-000000000000'). You can define your classes that derive from these classes (since they are part of the global namespace) and you can declare Instance Nodes whose OID matches one of the ET Class definitions.

This section describes the set of ET Class Definitions. There are currently three simple Class Definitions defined in the global product namespace:

- ET_GenericNode (OID=0x80000001) – this is a simple generic node that has no properties.
- ET_CollectionNode (OID=0x80000002) – this node type also has no properties. This node can be used when you need to declare a generic collection node in a

hierarchy of node objects. For example, you may declare an Instance Node named 'Managed Processes' of this type which represents a collection of managed processes nodes. The child nodes would presumably have some node type that you define in your product namespace, but the collection node itself is generic. Keep in mind, that you are under no obligation to use this generic class. You can define your own generic collection node that may have additional properties.

- ET_UnknownNodeType (OID=0x80000003) – this generic node type is used for parent Instance Nodes when you declare an Instance Node whose parent Instance Node does not yet exist. Normally, you want to make sure that all parent Instance Nodes are already defined when you declare a new Instance Node. If this condition is not satisfied, the ET framework will create a generic parent Instance Node of this type for you. You can then go back and change the node type using the <FixInstanceNode> message. See section 1.18.3 for more details about this.

Note that the OID values for nodes in the global namespace all have the high order bit (bit 31) set. If you recall from section 1.8.2, you can choose any unique 32-bit integer for OIDs in your product namespace as long as they are in the range [0-0x7fffffff].

The following XML document describes the set of Class Definitions for the global ET product namespace.

```
<p:ListenerProductAdmin
xmlns:p='urn:www.cisco.com:ListenerProductAdmin'
ETVersion='3.0'>

   <RegisterProduct ProductNamespaceGUID='00000000-0000-0000-0000-
000000000000' BaselineVersion='0.0' NewVersion='1.0' Name='ET'
Manufacturer='Cisco Systems, Inc.'>
     <Description>ET</Description>

     <ClassDefinition OID="2147483649">
       <Name>ET_GenericNode</Name>
       <Description>Generic node.</Description>
     </ClassDefinition>

     <ClassDefinition OID="2147483650">
       <Name>ET_CollectionNode</Name>
       <Description>Generic collection node with no
properties.</Description>
     </ClassDefinition>

     <ClassDefinition OID="2147483651">
       <Name>ET_UnknownNodeType</Name>
       <Description>Unknown node type.</Description>
     </ClassDefinition>

   </RegisterProduct>
```

```
</p:ListenerProductAdmin>
```

Appendix A
Error Messages

<RegisterProduct> Status Values		
Value	**Mnemonic**	**Error Text**
0x80200001 (-2145386495)	ET_E_INVALIDSCHEMA	Invalid schema.
0x80200002 (-2145386494)	ET_E_INSUFFICIENTPRIVILEGE	Insufficient privileges to execute this command.
0x80200003 (-2145386493)	ET_E_ERRORSCONTAINED	Some errors exist in the child elements.
0x80220001 (-2145255423)	MSGMGR_E_KEYALREADYEXISTS	%1 is not unique. It has already been assigned to %2.
0x80220002 (-2145255422)	MSGMGR_E_PRODUCTNAMEALREADYEXISTS	The Product Name is not unique.
0x80220003 (-2145255421)	MSGMGR_E_KEYDOESNOTEXIST	%1 does not exist.
0x80220004 (-2145255420)	MSGMGR_E_NAMEALREADYEXISTS	%1 is not unique. It has already been assigned.
0x00220005 (2228229)	MSGMGR_S_OLDVERSION	The Version number supplied is less than what is currently registered.
0x00220006 (2228230)	MSGMGR_S_DUPLICATEPRODUCTNAME	The product name conflicts with an existing product name. The product name will be further qualified by its manufacturer.
0x80220007 (-2145255417)	MSGMGR_E_SITEALREADYONLINE	The Site is already online.
0x80220008 (-2145255416)	MSGMGR_E_SITEALREADYOFFLINE	The Site is already offline.
0x80220009 (-2145255415)	MSGMGR_E_UNKNOWNPRODUCTVERSION	The product BaselineVersion is not known.

0x8022000A (-2145255414)	MSGMGR_E_PRODUCTVERSIONMISMATCH	The product Version number does not match what is currently registered (%1).
0x0022000B (228235)	MSGMGR_S_UNDECLAREDINSTANCENODES	Undeclared parent nodes. The instance nodes %1 for this node have not been defined. Generic instance nodes have been created.
0x8022000C (-2145255412)	MSGMGR_E_INVALIDNODETYPE	The required Instance Node OID is not the correct type for this operation.
0x8022000D (-2145255411)	MSGMGR_E_INVALIDVALUETYPE	The data value passed could not be coerced to the proper type.
0x8022000E (-2145255410)	MSGMGR_E_INVALIDPRODUCTNAMESPACE	The ProductNamespaceGUID specified is invalid in this context.
0x8022000F (-2145255409)	MSGMGR_E_INVALIDHEARTBEATSOURCE	The Source specified in the Heartbeat message does not match what was sent in the SiteOnline message sent earlier.

Appendix B
Language Identifiers

A language identifier is a standard international numeric abbreviation for a country or geographical region. Each language has a unique language identifier, a 16-bit value that consists of a primary language identifier and a sublanguage identifier. The following illustration shows the format of the bits in this Language Identifier.

Bits															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SubLanguage ID						Primary Language ID									

The following table shows the defined Primary Language IDs.

Identifier	Language
0x00	Neutral
0x01	Arabic
0x02	Bulgarian
0x03	Catalan
0x04	Chinese
0x05	Czech
0x06	Danish
0x07	German
0x08	Greek
0x09	English
0x0a	Spanish
0x0b	Finnish
0x0c	French
0x0d	Hebrew
0x0e	Hungarian
0x0f	Icelandic
0x10	Italian
0x11	Japanese
0x12	Korean
0x13	Dutch
0x14	Norwegian
0x15	Polish
0x16	Portuguese
0x18	Romanian
0x19	Russian
0x1a	Croatian
0x1a	Serbian
0x1b	Slovak
0x1c	Albanian
0x1d	Swedish
0x1e	Thai
0x1f	Turkish

0x20	Urdu
0x21	Indonesian
0x22	Ukrainian
0x23	Belarusian
0x24	Slovenian
0x25	Estonian
0x26	Latvian
0x27	Lithuanian
0x29	Farsi
0x2a	Vietnamese
0x2b	Armenian
0x2c	Azeri
0x2d	Basque
0x2f	FYRO Macedonian
0x36	Afrikaans
0x37	Georgian
0x38	Faeroese
0x39	Hindi
0x3e	Malay
0x3f	Kazak
0x40	Kyrgyz
0x41	Swahili
0x43	Uzbek
0x44	Tatar
0x45	Bengali
0x46	Punjabi
0x47	Gujarati
0x48	Oriya
0x49	Tamil
0x4a	Telugu
0x4b	Kannada
0x4c	Malayalam
0x4d	Assamese
0x4e	Marathi
0x4f	Sanskrit
0x50	Mongolian
0x56	Galician
0x57	Konkani
0x58	Manipuri
0x59	Sindhi
0x5a	Syriac
0x60	Kashmiri
0x61	Nepali
0x65	Divehi
0x7f	

The following table shows the defined SubLanguage IDs:

0x00	Language neutral
0x01	User Default
0x02	System Default
0x01	Arabic (Saudi Arabia)
0x02	Arabic (Iraq)

0x03	Arabic (Egypt)
0x04	Arabic (Libya)
0x05	Arabic (Algeria)
0x06	Arabic (Morocco)
0x07	Arabic (Tunisia)
0x08	Arabic (Oman)
0x09	Arabic (Yemen)
0x0a	Arabic (Syria)
0x0b	Arabic (Jordan)
0x0c	Arabic (Lebanon)
0x0d	Arabic (Kuwait)
0x0e	Arabic (U.A.E.)
0x0f	Arabic (Bahrain)
0x10	Arabic (Qatar)
0x01	Azeri (Latin)
0x02	Azeri (Cyrillic)
0x01	Chinese (Traditional)
0x02	Chinese (Simplified)
0x03	Chinese (Hong Kong SAR, PRC)
0x04	Chinese (Singapore)
0x05	Chinese (Macau SAR)
0x01	Dutch
0x02	Dutch (Belgian)
0x01	English (US)
0x02	English (UK)
0x03	English (Australian)
0x04	English (Canadian)
0x05	English (New Zealand)
0x06	English (Ireland)
0x07	English (South Africa)
0x08	English (Jamaica)
0x09	English (Caribbean)
0x0a	English (Belize)
0x0b	English (Trinidad)
0x0c	English (Zimbabwe)
0x0d	English (Philippines)
0x01	French
0x02	French (Belgian)
0x03	French (Canadian)
0x04	French (Swiss)
0x05	French (Luxembourg)
0x06	French (Monaco)
0x01	German
0x02	German (Swiss)
0x03	German (Austrian)
0x04	German (Luxembourg)
0x05	German (Liechtenstein)
0x01	Italian
0x02	Italian (Swiss)
0x02	Kashmiri
0x01	Korean
0x01	Lithuanian

0x01	Malay (Malaysia)
0x02	Malay (Brunei Darassalam)
0x02	Nepali (India)
0x01	Norwegian (Bokmal)
0x02	Norwegian (Nynorsk)
0x01	Portuguese (Brazil)
0x02	Portuguese
0x02	Serbian (Latin)
0x03	Serbian (Cyrillic)
0x01	Spanish (Castilian)
0x02	Spanish (Mexican)
0x03	Spanish (Modern)
0x04	Spanish (Guatemala)
0x05	Spanish (Costa Rica)
0x06	Spanish (Panama)
0x07	Spanish (Dominican Republic)
0x08	Spanish (Venezuela)
0x09	Spanish (Colombia)
0x0a	Spanish (Peru)
0x0b	Spanish (Argentina)
0x0c	Spanish (Ecuador)
0x0d	Spanish (Chile)
0x0e	Spanish (Uruguay)
0x0f	Spanish (Paraguay)
0x10	Spanish (Bolivia)
0x11	Spanish (El Salvador)
0x12	Spanish (Honduras)
0x13	Spanish (Nicaragua)
0x14	Spanish (Puerto Rico)
0x01	Swedish
0x02	Swedish (Finland)
0x01	Urdu (Pakistan)
0x02	Urdu (India)
0x01	Uzbek (Latin)
0x02	Uzbek (Cyrillic)

Finally, putting these together, we get the set of the full 16-bit Language Identifiers that are used in the Language ID fields in the ET specifications:

Identifier	Language
0x0000	Language Neutral
0x007f	The language for the invariant locale (LOCALE_INVARIANT).
0x0400	Process or User Default Language
0x0800	System Default Language
0x0436	Afrikaans
0x041c	Albanian
0x0401	Arabic (Saudi Arabia)
0x0801	Arabic (Iraq)
0x0c01	Arabic (Egypt)
0x1001	Arabic (Libya)
0x1401	Arabic (Algeria)
0x1801	Arabic (Morocco)
0x1c01	Arabic (Tunisia)

0x2001	Arabic (Oman)
0x2401	Arabic (Yemen)
0x2801	Arabic (Syria)
0x2c01	Arabic (Jordan)
0x3001	Arabic (Lebanon)
0x3401	Arabic (Kuwait)
0x3801	Arabic (U.A.E.)
0x3c01	Arabic (Bahrain)
0x4001	Arabic (Qatar)
0x042b	**Windows 2000 or later:** Armenian. This is Unicode only.
0x042c	Azeri (Latin)
0x082c	Azeri (Cyrillic)
0x042d	Basque
0x0423	Belarussian
0x0402	Bulgarian
0x0455	Burmese
0x0403	Catalan
0x0404	Chinese (Taiwan)
0x0804	Chinese (PRC)
0x0c04	Chinese (Hong Kong SAR, PRC)
0x1004	Chinese (Singapore)
0x1404	**Windows 98/Me, Windows 2000 or later:** Chinese (Macau SAR)
0x041a	Croatian
0x0405	Czech
0x0406	Danish
0x0465	**Whistler:** Divehi. This is Unicode only.
0x0413	Dutch (Netherlands)
0x0813	Dutch (Belgium)
0x0409	English (United States)
0x0809	English (United Kingdom)
0x0c09	English (Australian)
0x1009	English (Canadian)
0x1409	English (New Zealand)
0x1809	English (Ireland)
0x1c09	English (South Africa)
0x2009	English (Jamaica)
0x2409	English (Caribbean)
0x2809	English (Belize)
0x2c09	English (Trinidad)
0x3009	**Windows 98/Me, Windows 2000 or later:** English (Zimbabwe)
0x3409	**Windows 98/Me, Windows 2000 or later:** English (Philippines)
0x0425	Estonian
0x0438	Faeroese
0x0429	Farsi
0x040b	Finnish
0x040c	French (Standard)
0x080c	French (Belgian)
0x0c0c	French (Canadian)
0x100c	French (Switzerland)
0x140c	French (Luxembourg)
0x180c	**Windows 98/Me, Windows 2000 or later:** French (Monaco)
0x0456	**Whistler:** Galician

0x0437	**Windows 2000 and later:** Georgian. This is Unicode only.
0x0407	German (Standard)
0x0807	German (Switzerland)
0x0c07	German (Austria)
0x1007	German (Luxembourg)
0x1407	German (Liechtenstein)
0x0408	Greek
0x0447	**Whistler:** Gujarati. This is Unicode only.
0x040d	Hebrew
0x0439	**Windows 2000 and later:** Hindi. This is Unicode only.
0x040e	Hungarian
0x040f	Icelandic
0x0421	Indonesian
0x0410	Italian (Standard)
0x0810	Italian (Switzerland)
0x0411	Japanese
0x044b	**Whistler:** Kannada. This is Unicode only.
0x0860	Kashmiri
0x043f	Kazakh
0x0457	**Windows 2000 and later:** Konkani. This is Unicode only.
0x0412	Korean
0x0812	**Windows 95, Windows NT 4.0 only:** Korean (Johab)
0x0440	**Whistler:** Kyrgyz.
0x0426	Latvian
0x0427	Lithuanian
0x0827	**Windows 98 only:** Lithuanian (Classic)
0x042f	FYRO Macedonian
0x043e	Malay (Malaysian)
0x083e	Malay (Brunei Darussalam)
0x0458	Manipuri
0x044e	**Windows 2000 and later:** Marathi. This is Unicode only.
0x0450	**Whistler:** Mongolian
0x0414	Norwegian (Bokmal)
0x0814	Norwegian (Nynorsk)
0x0415	Polish
0x0416	Portuguese (Brazil)
0x0816	Portuguese (Portugal)
0x0446	**Whistler:** Punjabi. This is Unicode only.
0x0418	Romanian
0x0419	Russian
0x044f	**Windows 2000 and later:** Sanskrit. This is Unicode only.
0x0c1a	Serbian (Cyrillic)
0x081a	Serbian (Latin)
0x0459	Sindhi
0x041b	Slovak
0x0424	Slovenian
0x040a	Spanish (Traditional Sort)
0x080a	Spanish (Mexican)
0x0c0a	Spanish (Modern Sort)
0x100a	Spanish (Guatemala)
0x140a	Spanish (Costa Rica)
0x180a	Spanish (Panama)

0x1c0a	Spanish (Dominican Republic)
0x200a	Spanish (Venezuela)
0x240a	Spanish (Colombia)
0x280a	Spanish (Peru)
0x2c0a	Spanish (Argentina)
0x300a	Spanish (Ecuador)
0x340a	Spanish (Chile)
0x380a	Spanish (Uruguay)
0x3c0a	Spanish (Paraguay)
0x400a	Spanish (Bolivia)
0x440a	Spanish (El Salvador)
0x480a	Spanish (Honduras)
0x4c0a	Spanish (Nicaragua)
0x500a	Spanish (Puerto Rico)
0x0430	Sutu
0x0441	Swahili (Kenya)
0x041d	Swedish
0x081d	Swedish (Finland)
0x045a	**Whistler:** Syriac. This is Unicode only.
0x0449	**Windows 2000 and later:** Tamil. This is Unicode only.
0x0444	Tatar (Tatarstan)
0x044a	**Whistler:** Telugu. This is Unicode only.
0x041e	Thai
0x041f	Turkish
0x0422	Ukrainian
0x0420	**Windows 98/Me, Windows 2000 or later:** Urdu (Pakistan)
0x0820	Urdu (India)
0x0443	Uzbek (Latin)
0x0843	Uzbek (Cyrillic)
0x042a	**Windows 98/Me, Windows NT 4.0 and later:** Vietnamese

5

End of Appendix A